

Learning 3D LiDAR Object Detection without Human Annotations

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät
der Eberhard Karls Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von
Stefan Andreas Baur
aus Waiblingen

Tübingen
2024

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der
Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:

08.05.2025

Dekan:

Prof. Dr. Thilo Stehle

1. Berichterstatter/-in:

Prof. Dr. Andreas Geiger

2. Berichterstatter/-in:

Prof. Dr. MarkusENZweiler

Abstract

The emergence of [Deep Learning \(DL\)](#) has revolutionized the field of computer vision, by allowing machines to outperform humans across a wide range of tasks. In particular in the domain of [Autonomous Driving \(AD\)](#) and [Advanced Driver Assistance Systems \(ADAS\)](#), [DL](#) has been successfully applied to a multitude of tasks, such as object detection, segmentation, tracking, prediction, trajectory planning and various others. To guarantee optimal performance, safety and reliability of [AD](#) systems, [DL](#) methods have to be trained and tested rigorously on large and diverse datasets. This presents a challenge, due to the complexity and variability of real-world environments, as well as the costs associated with acquiring sufficient data of rare but critical events.

One way to address this issue lies in simulation, which can generate large volumes of synthetic data at a fraction of the cost of real-world data acquisition. Furthermore, simulation provides the benefit of allowing complete and independent control over all influences on the system, allowing for in-depth analysis of the system’s behavior w.r.t. to changes in the environment, without endangering human lives or expensive equipment. In the past, simulation of realistic [Light Detection and Ranging \(LiDAR\)](#) data has not received as much attention as simulation of photorealistic images. Therefore, we present a physically-based [LiDAR](#) simulation model that is capable of generating realistic [LiDAR](#) data, and use it to train and evaluate various [DL](#) models for [LiDAR](#) object detection, comparing our method against other strong baselines. This represents the first major contribution of this thesis.

One of the most significant challenges in simulation is attaining a level of realism that ensures that methods perform consistently in both simulated and real-world environments, allowing for transfer of insights from simulation to the real world. An alternative, equally cost-efficient approach to obtain large amounts of annotated data is to employ self-supervised learning on data from the target domain directly, eliminating the risk of unrealistic data such as simulation artifacts. Self-supervised learning exploits the inherent structure of the data to produce supervision signals for training, without the need for any human annotation. Therefore, in this thesis we propose a novel self-supervised learning method that learns per-point motion estimation from raw [LiDAR](#) data sequences only. Additionally, we demonstrate that our method is able to segment moving objects in the scene and use this information to improve the motion estimation task. Our method is evaluated on three different datasets and outperforms our strong baselines. This novel method marks the second major contribution of this thesis.

Motion cues are a powerful source of information for [AD](#) systems. Essentially, all moving objects in the scene are of interest to the [AD](#) system, as they have the potential to directly influence the vehicle’s trajectory. However, this potential extends beyond *moving* objects, and includes those that have the *capability* to move, i.e. all *movable* objects. Using

Abstract

the self-supervised per-point motion estimation mentioned above, we propose a novel method that iteratively discovers all *movable* objects in the scene. This enables us to train state-of-the-art **LiDAR** object detectors without the need for any human annotation, which is especially useful in **AD**, where obtaining labeled data is expensive and time-consuming. We compare our method against several other strong baselines and demonstrate that our method improves over the state-of-the-art on four different datasets, using two different object detectors and identical hyperparameters for all settings, which represents the final major contribution of this thesis.

Kurzfassung

Das Aufkommen von [Deep Learning \(DL\)](#) hat das Feld der Computer Vision revolutioniert, indem es Maschinen in die Lage versetzt hat, Menschen in einer Vielzahl von Aufgaben zu übertreffen. Insbesondere im Bereich des autonomen Fahrens ([AD](#)) und der Fahrerassistenzsysteme ([ADAS](#)) wird [DL](#) erfolgreich für zahlreiche Aufgaben angewendet, wie Objekterkennung, Segmentierung, Vorhersage, Trajektorienplanung und viele andere. Um optimale Leistung, Sicherheit und Zuverlässigkeit von [AD](#)-Systemen zu gewährleisten, müssen [DL](#)-Methoden rigoros auf großen und abwechslungsreichen Datensätzen trainiert und getestet werden. Dies stellt eine Herausforderung dar, aufgrund der Komplexität und erforderlichen Variabilität der Datensätze sowie der Kosten, die mit der Erfassung ausreichender Daten seltener, aber kritischer Ereignisse verbunden sind.

Eine Möglichkeit, dieses Problem zu adressieren, ist die Simulation, die große Mengen synthetischer Daten zu einem Bruchteil der Kosten der realen Datenerfassung erzeugen kann. Außerdem hat sie den Vorteil, dass alle Systemeinflüsse vollständig und unabhängig kontrolliert werden können, was eine eingehende Analyse des Systemverhaltens in Bezug auf Änderungen in der Umgebung ermöglicht, ohne Menschenleben oder teure Prototypen zu gefährden. In der Vergangenheit hat die Simulation realistischer [Light Detection and Ranging \(LiDAR\)](#)-Daten weniger Aufmerksamkeit erhalten als die Simulation fotorealistischer Bilder. Daher präsentieren wir ein physikalisch basiertes [LiDAR](#)-Simulationsmodell, das in der Lage ist, realistische [LiDAR](#)-Daten zu erzeugen, und verwenden es, um verschiedene [DL](#)-Modelle für die [LiDAR](#)-Objekterkennung zu trainieren und zu evaluieren, wobei wir unsere Methode mit anderen starken Baselines vergleichen. Dies stellt den ersten wesentlichen Beitrag dieser Arbeit dar.

Eine der größten Herausforderungen in der Simulation besteht darin, ein Maß an Realismus zu erreichen, das sicherstellt, dass Methoden sowohl in simulierten als auch in realen Umgebungen konsistent funktionieren, sodass Erkenntnisse aus der Simulation in die reale Welt übertragen werden können. Ein alternativer, ebenso kosteneffizienter Ansatz zur Gewinnung großer Mengen annotierter Daten besteht darin, selbstüberwachtes Lernen direkt auf Daten aus der Zielumgebung anzuwenden, wodurch das Risiko unrealistischer Daten (zum Beispiel durch Simulationsartefakte) verhindert wird. Selbstüberwachtes Lernen nutzt die inhärente Struktur der Daten, um Trainingssignale zu erzeugen, ohne dass menschliche Annotationen erforderlich sind. Daher schlagen wir in dieser Arbeit eine neuartige Methode des selbstüberwachten Lernens vor, die die Bewegungsschätzung pro Punkt nur aus Roh-[LiDAR](#)-Datensequenzen lernt. Zusätzlich zeigen wir, dass unsere Methode in der Lage ist, sich bewegende Objekte in der Szene zu segmentieren und diese Informationen zu nutzen, um die Bewegungsschätzung zu verbessern. Unsere Methode wird auf drei verschiedenen Datensätzen evaluiert und übertrifft unsere starken Baselines. Diese neuartige Methode stellt den zweiten wesentlichen Beitrag dieser Arbeit dar.

Bewegungsinformationen sind eine mächtige Informationsquelle für AD-Systeme. Im Wesentlichen sind alle sich bewegenden Objekte in der Szene für das AD-System relevant, da sie das Potenzial haben, die Trajektorie des Fahrzeugs direkt zu beeinflussen. Dieses Potenzial erstreckt sich jedoch über *bewegte* Objekte hinaus und umfasst auch solche, die die Fähigkeit zur Bewegung haben, d.h. alle *beweglichen* Objekte. Unter Verwendung der oben genannten selbstüberwachten punktwisen Bewegungsschätzung schlagen wir eine neuartige Methode vor, die iterativ alle *beweglichen* Objekte in der Szene entdeckt. Dies ermöglicht es uns, LiDAR-Objektdetektoren ohne menschliche Annotationen zu trainieren, was besonders im AD Bereich nützlich ist, wo die Gewinnung von gelabelten Daten teuer und zeitaufwendig ist. Wir vergleichen unsere Methode mit mehreren anderen starken Baselines und zeigen, dass unsere Methode mit identischen Hyperparamtern den Stand der Technik auf vier verschiedenen Datensätzen für zwei verschiedene Objektdetektoren verbessert. Dies stellt den letzten wesentlichen Beitrag dieser Dissertation dar.

Acknowledgments

I owe much to a group of exceptional individuals who have provided guidance, support, joy and funding along the way.

Firstly, I want to express my sincere gratitude to Prof. Dr.-Ing. Andreas Geiger for giving me the chance to pursue my research under his guidance. His support, patience, and the inclusive atmosphere in his research group are unparalleled. The inspiring research retreats are some of the most fun memories I have from my time as a PhD student. I am thankful to Prof. Dr. rer. nat. Markus Enzweiler for his encouragement to pursue a PhD, which set this journey in motion. Thank you also for acting as second reviewer of this dissertation. I am further thankful to Dr.-Ing. Frank Moosmann, who has been a source of constant support and guidance, especially during the more challenging periods of the PhD. His efforts and dedication, and focus, especially during the manuscript writing for our publications, have been greatly appreciated.

My gratitude extends to my first and second bosses at Mercedes-Benz, Roland Schweiger and Thomas Wohlgemuth. Their support for my PhD ambitions and the freedom they provided in my work were crucial to my development and well-being during this time. I also want to thank my third boss, Frank Schuster, for his enthusiasm about my work and his continuous encouragement to finish my PhD. A big thank you goes to Vitali Kaiser, Robert Ulbrich, Fabian Poggenhans, Samuel Vogelmann, Zeeshan Shareef, Simon Rühle, and everyone else involved in providing a stable and reliable computing platform, which is the backbone of our research. Their hard work is essential for all PhD students and has not gone unnoticed.

As to my PhD student colleagues, I am particularly grateful to David Emmerichs for sharing his expertise in physics and mathematics with me. His problem-solving skills and willingness to help others are inspiring. Niklas Hanselmann deserves thanks for his encouragement and levelheadedness when handling difficult reviewer feedback during the peer review process. I also want to thank Christoph Rist for introducing me to Blender and for his support in creating renderings for my publications. Larissa Triess, thank you for our fruitful exchange of ideas on sim2real domain transfer. I also want to acknowledge Robin Heinzler and Mariella Dreissig for the engaging discussions we've had. Thanks to Philipp Schindler for a comprehensive introduction to the physics of LiDAR sensors.

To Jasmin, your understanding and support have been a cornerstone of my ability to pursue this degree. Your encouragement, patience and mental support have been crucial, especially during the tough times. And to my family, knowing I had your backing has been a great comfort.

To everyone who has been a part of this journey, thank you. Your contributions have been essential to the completion of my PhD.

Contents

Abstract	iii
Kurzfassung	v
Acknowledgments	vii
Notation	xiii
List of Symbols	xv
List of Abbreviations	xvii
1 Introduction	1
1.1 Problem Statement	1
1.2 Contributions	2
1.3 Thesis Outline	3
2 Background	5
2.1 Time-of-Flight: Working Principle of LiDAR	5
2.2 Point Clouds and Range Images	6
2.3 Lidar Scene Flow	8
2.3.1 Kabsch Algorithm	9
2.3.2 Metrics	11
2.4 LiDAR 3D Object Detection	12
2.5 Unsupervised Learning	13
2.6 Self-Supervised Learning	14
2.7 Self Training	15
2.8 Datasets for AD	15
2.8.1 KITTI	16
2.8.2 nuScenes	16
2.8.3 Waymo Open Dataset(WOD)	16
2.8.4 Argoverse 2(AV2)	17
3 Related Work	19
3.1 LiDAR Simulation	19
3.1.1 Conventional Model-based LiDAR Simulation	19
3.1.2 Sim2Real Domain Adaption & Data-driven, Generative Simulation	21

3.2	Scene Flow and Motion Segmentation	22
3.2.1	Motion Segmentation	22
3.2.2	Supervised Point-Cloud-Based Scene Flow	23
3.2.3	Self-Supervised Learning	23
3.3	LiDAR Object Detection	25
3.3.1	Single Frame LiDAR 3D Object Detection	25
3.3.2	Object Distilling from Motion Cues	25
3.3.3	Pseudo Ground Truth for Object Detection	26
4	Physically-based LiDAR Simulation	27
4.1	Introduction	27
4.2	Working principle of LiDAR	29
4.3	Physical Model	32
4.3.1	Model Overview	33
4.3.2	Time-Surface Model	36
4.3.3	Surface-Reflection Model	39
4.3.4	Threshold-Intersection Model	42
4.3.5	Noise Model	44
4.3.6	Influence of Laser Layout on Raydrops	46
4.4	Parameter Selection	52
4.5	Implementation Details	52
4.6	Evaluation	55
4.6.1	Baselines	56
4.6.2	Visual Comparison with Real-World Data	56
4.6.3	Statistical Comparison with Real-World Data	62
4.6.4	Downstream Task: LiDAR Object Detection	69
4.6.5	Measuring Suitability for Validation	74
4.7	Conclusion	75
4.7.1	Assumptions and Resulting Limitations	75
4.7.2	Summary	76
4.7.3	Outlook	78
5	Self-Supervised LiDAR Scene Flow and Motion Segmentation	79
5.1	Introduction	79
5.2	Method	81
5.2.1	Network Architecture	81
5.2.2	Loss	84
5.3	Evaluation	86
5.3.1	Experimental Setup	86
5.3.2	Impact of Point Cloud Density	88
5.3.3	Training Dataset and Domain Shift	88
5.3.4	Evaluation on LiDAR Data	89
5.3.5	Supervised Flow Prediction	89
5.3.6	Ablation Study	91

5.3.7	Motion Segmentation	92
5.4	Conclusion	93
5.5	Outlook	93
6	LiDAR-only Self-Supervised 3D Object Detection	95
6.1	Introduction	95
6.2	Method	97
6.2.1	Preprocessing	97
6.2.2	Initial Pseudo Ground Truth Generation	98
6.2.3	Trajectory-Regularized Self-Training	100
6.3	Evaluation	101
6.3.1	Datasets and Metrics	102
6.3.2	Networks	103
6.3.3	Baselines	103
6.3.4	Results	104
6.3.5	Ablation Study	105
6.4	Conclusion	107
7	Conclusion	109
A	Publications	111
B	Appendix: Credits	113
B.1	SLIM: Self-Supervised LiDAR Scene Flow and Motion Segmentation	113
B.2	LISO: Lidar-only Self-Supervised 3D Object Detection	113
C	Appendix: Self-Supervised LiDAR Scene Flow and Motion Segmentation	115
C.1	Computation of Confidence Weights	115
C.2	Moving Classification Threshold	116
C.3	Robust Outlier Metric	118
C.4	Classification	119
C.5	Architecture & Parameter Details	120
C.5.1	Training	120
C.5.2	PillarFeatureNet	121
C.5.3	RAFT	122
C.6	Transferability of the SLIM Loss Framework onto PointPWC	122
C.7	Runtimes	123
C.8	Failure Cases	124
D	Appendix: LiDAR-only Self-Supervised 3D Object Detection	125
D.1	Implementation Details	125
D.2	Self-supervised LiDAR Scene Flow	125
D.3	Additional Ablations	126
D.4	Performance of LiDAR Scene Flow Clustering on nuScenes	126
D.5	Quality of Pseudo Ground Truth during Iterative Self-Training	128

Contents

D.6 Qualitative Results	129
D.7 Quantitative Results	129
Bibliography	133

Notation

General

Scalars	Regular lower case	x, y, z
Functions	Regular upper and lower case	f, g, h, F, G, H
Vectors	Bold lower case	$\mathbf{x}, \mathbf{y}, \mathbf{z}$
Matrices	Bold upper case	$\mathbf{X}, \mathbf{Y}$
Distributions	Calligraphic upper case	$\mathcal{U}(\cdot), \mathcal{N}(\cdot)$
Sets	Calligraphic upper case	$\mathcal{A}, \mathcal{B}$

Spaces

$\mathbb{R}$	Real numbers
$\mathbb{N}$	Natural numbers
$\mathbb{N}^+$	Natural numbers (greater than 0)

Probability

$\mathcal{U}(\cdot)$	Uniform Distribution
$\mathcal{N}(\cdot)$	Gaussian Distributions
$p(\cdot)$	Probability

Geometry

$\mathbf{p}$	A point in 3D, $\mathbf{p} \in \mathbb{R}^3$
$\mathbf{f}$	A scene flow vector in 3D, $\mathbf{f} \in \mathbb{R}^3$
$\mathcal{P}$	A point cloud with n points in 3D, $\mathcal{P} \in \mathbb{R}^{n \times 3}$
$\mathcal{F}$	A scene flow field with n flow vectors in 3D, $\mathcal{F} \in \mathbb{R}^{n \times 3}$

Notation

Others

$\frac{df}{dt}$	Total derivative of f w.r.t. t
$\frac{\partial f}{\partial t}$	Partial derivative of f w.r.t. t
$\ \cdot\ _p$	L^p Norm
$\mathbf{x}^T, \mathbf{X}^T$	Vector, matrix transpose
$\mathbf{x}^{(i)}$	Component notation, i-th element of vector $\mathbf{x}$ ($i \in \mathbb{N}$)
$\mathbf{X}^{(i,j)}$	Component notation, i-th, j-th element of matrix $\mathbf{X}$ ($i, j \in \mathbb{N}$)
$\mathbf{I}_n$	$n \times n$ identity matrix
$H(x)$	Heaviside function
$\mathbb{I}[\cdot]$	Indicator function
$\hat{x}$	Estimated value of x

List of Symbols

Φ	All of the parameters of the LiDAR system
Φ_{P_E}	Emitted power by the laser
$\Phi_{P_{E,jitter}}$	Amplitude jitter of the emitted laser power
$\Phi_{\sigma_{P_E}}$	Standard deviation of the power emitted by laser emitter
Φ_{P_x}	Minimum power of a detectable signal (signal visibility threshold of the receiver)
$\Phi_{t_{pw}}$	Pulse width (time) of the laser pulse
Φ_{D_R}	Aperture diameter of the laser receiver
Φ_{D_L}	Laser beam diameter at the emitter
Φ_{ψ}	Laser beam divergence of the emitted laser ray
Φ_{SNR}	Signal-to-noise ratio (SNR) of the Avalanche Photo Diode (APD) of the LiDAR receiver
R	Range to the measured target
θ	Elevation angle of the target relative to the sensor
ϕ	Azimuth angle of the target relative to the sensor
w	Laser beam width at the target
A_{lf}	Laser beam footprint at the target
σ	Backscattering cross-section of the laser beam at the target
f_r	Bidirectional reflectance distribution function
$\mathbf{f}_r$	Bidirectional reflectance distribution function (vector)
Θ	All of the target surface parameters
Θ_m	Metallicness of the target surface
Θ_{f_0}	Base reflectivity
Θ_{rgb}	Matte RGB base color of the target surface
Θ_r	Roughness of the target surface
Θ_n	Surface normal of the target surface (in sensor coordinates)
Θ_{θ}	Angle of incidence of the laser ray on the target surface
P_R	Total received power by receiver (varies over time)
t	Time
t_x	Time at which the power at the receiver P_R crosses the threshold Φ_{P_x}

List of Symbols

$\hat{t}$	Measured time of intersection
$t_{\text{res,TDC}}$	Time resolution of the Time Digital Converter (TDC)
$t_{\text{e, Discretization}}$	Time error due to discretization error (TDC resolution)
σ_{clock}	Standard deviation of the clock time error
$t_{\text{e,Clock}}$	Clock time error
$\sigma_{\text{timing jitter}}$	Standard deviation of the time error caused by the amplitude jitter
$\sigma_{\text{signal jitter}}$	Standard deviation of received signal jitter
$t_{\text{e, Amplitude}}$	Time jitter due to received signal jitter
t_{e}	Total time error of the measurement
$\hat{R}$	Measured Distance to the target
β	Atmospheric damping coefficient
Ω_s	Solid Angle of the receiving aperture at the target
c	Speed of light
τ	Atmospheric damping for a given range and atmospheric damping coefficient β
$\mathbf{v}$	View vector (in sensor coordinates)
$\mathbf{h}$	Half vector (in sensor coordinates)

List of Abbreviations

AD	Autonomous Driving
AV	Autonomous Vehicle
ADAS	Advanced Driver Assistance Systems
AEE	Average Endpoint Error
AP	Average Precision
BEV	Bird’s Eye View
BlenSor	Blender Sensor Simulation Toolbox
BRDF	Bidirectional Reflectance Distribution Function
CNN	Convolutional Neural Network
DL	Deep Learning
DSP	Digital Signal Processor
EE	Endpoint Error
FOV	Field Of View
GAN	Generative Adversarial Network
GPS	Global Positioning System
HDL64	Velodyne LiDAR sensor type “HDL64”
IMU	Inertial Measurement Unit
IoU	Intersection over Union
KITTI-SF	KITTI Stereo Flow
k-NN	K-Nearest Neighbors
laser	Light Amplification by Stimulated Emission of Radiation
LE	Laser Emitter
LD	Laser Diode
LiDAR	Light Detection and Ranging
LR	Laser Receiver
mAP	Mean Average Precision
mIoU	Mean Intersection over Union
NeRF	Neural Radiance Field
NMS	Non-Maximum Suppression
PD	Photo Diode
PFN	Pillar Feature Net
PP	PointPillars
RADAR	Radio Detection and Ranging
RoI	Region of Interest
SLAM	Simultaneous Localization and Mapping

List of Abbreviations

SNR	Signal-to-Noise Ratio
TDC	Time Digital Converter
TIA	Transimpedance Amplifier

List of Figures

2.1	Time-of-flight Principle	6
2.2	Point Cloud	7
2.3	Range Image	8
2.4	Scene Flow	9
2.5	Example Scene Flow on KITTI	10
2.6	Lidar 3D Object Detection	12
3.1	BlenSor Visibility	21
4.1	Simplified LiDAR Echo	30
4.2	Basic LiDAR Design	31
4.3	Simulation Overview	35
4.4	Beam Geometry and Illuminated Area over Time	38
4.5	Walk Error	44
4.6	Influence of Signal Jitter on Time Measurement	46
4.7	Example of Raydrops on Poles in the KITTI Dataset	47
4.8	HDL64 Laser Orientations	47
4.9	Empirical Probability of Raydrops in the KITTI Dataset	49
4.10	CARLA Range and RGB Images with Raydrops	50
4.11	KITTI Range and RGB Images with Raydrops	51
4.12	CARLA Hitboxes	53
4.13	CARLA Content	54
4.14	Range, SemanticKITTI Image	55
4.15	BlenSor Range Standard Deviation Comparison	57
4.16	Simulator Comparison Cars	57
4.17	Simulator Comparison Cars II	58
4.18	Simulator Comparison Road	59
4.19	Simulator Comparison Pedestrians	60
4.20	Simulator Comparison Pedestrian	61
4.21	Number of Points over Range	63
4.22	Number of Road Points over Range	64
4.23	Lidar Road Circles	65
4.24	Noise Comparison of Simulated and Real Data	68
4.25	Comparison Simulated and Real World	77
5.1	Scene Flow Prediction by SLIM	80
5.2	SLIM Network Overview	82
5.3	Qualitative Comparison of Scene Flow for PPWC, PF, and SLIM	86

List of Figures

5.4	Motion Segmentation by SLIM	92
6.1	3D Detections by LISO	96
6.2	Overview over LISO	97
6.3	Overview over LISO Preprocessing	98
6.4	Effect of Hyperparameters on Self-Learning	101
6.5	Qualitative Results on WOD by Oyster-CP and LISO-CP	105
6.6	Missed Ground Truth Boxes in the WOD Dataset	106
C.1	Influence of Threshold on AEE	117
C.2	Influence of Threshold on Scene Flow Metrics	118
C.3	Predicted and Ground Truth Motion Segmentation	121
C.4	Scene Flow Outliers	124
D.1	Clustering Performance Comparison	127
D.2	Precision and Recall of Pseudo Ground Truth Generated by Oyster and LISO	128
D.3	Comparison of Clustering Results for Oyster and LISO	129
D.4	Qualitative Comparison of Oyster and LISO on WOD	130

List of Tables

2.1	AD Datasets Comparison	16
4.1	Laser Echo Strength	32
4.2	Physical Simulation Parameters	52
4.3	Training Setup	70
4.4	Simulator Performance for LiDAR Object Detection	72
4.5	Simulator Performance for LiDAR Object Detection for Reduced Real-World Data	73
4.6	Detection Agreement Score	75
5.1	Influence of Point Cloud Density	89
5.2	SLIM Results with Domain Shift	90
5.3	SLIM Results on nuScenes	90
5.4	Scene Flow Experimental Results on CARLA	90
5.5	Scene Flow Experimental Results on KITTI-SF	91
5.6	Supervised Scene Flow Results	91
5.7	Ablation Study on nuScenes	92
6.1	Evaluation of LISO and Baselines on AV2 and nuScenes	102
6.2	Evaluation of LISO and Baselines on WOD	103
6.3	Evaluation of LISO and Baselines on KITTI	104
6.4	Ablation Study of LISO Components	106
C.1	Self-supervised Scene Flow Results on nuScenes	119
C.2	Classification Metrics on KITTI-SF	120
C.3	Ablation Study PPWC on nuScenes	123
C.4	Scene Flow Runtime Comparison	123
D.1	SLIM Scene Flow Metrics for LISO	126
D.2	Additional Ablations for LISO-CP on WOD	127
D.3	Full Evaluation of LISO and Baselines on nuScenes	131
D.4	Full Evaluation of LISO and Baselines on WOD	131

1 Introduction

Due to the ever-increasing demand for safer and more convenient mobility, [Advanced Driver Assistance Systems \(ADAS\)](#) are becoming more and more available on the market, with some lower-end systems such as lane keeping assistance or emergency braking assistance even becoming mandatory for new car models in certain jurisdictions [[Com21](#)]. On top of that, in the last years, the development of [Autonomous Driving \(AD\)](#) systems has made significant progress, with some companies offering autonomous taxi services in certain cities [[Way24](#)]. This increased level of automation requires reliable, robust and scalable environment perception systems for safe operation. In order to achieve a robust understanding of the scene, current vehicles are equipped with a multitude of sensors such as cameras, [Radio Detection and Ranging \(RADAR\)](#) or [Light Detection and Ranging \(LiDAR\)](#) sensors [[Pre20](#); [AG24](#)], which are constantly observing the environment. Sensor data is processed by a perception system, which nowadays employs [Deep Learning \(DL\)](#)-based methods due to their superior performance, accuracy and robustness in contrast to classical approaches. Their major drawback, however, is the requirement for large amounts of annotated data for training and validation, which is expensive to obtain. In addition, the necessary extensive testing of these systems is also expensive, as real-world testing is slow and labor-intensive, especially for rare edge cases that may have to be staged in controlled environments. Therefore, this thesis investigates possibilities how the cost of data annotation and testing can be reduced.

Section [1.1](#) starts with the problem statement and motivation. In Section [1.2](#), the main contributions of this thesis are introduced. Finally, Section [1.3](#) presents the structure of this thesis.

1.1 Problem Statement

Current [DL](#)-based, state-of-the-art perception systems for [AD](#) require large amounts of human-annotated data for training and evaluation. Especially when it comes to more complex types of annotations in the [LiDAR/3D](#) domain such as 3D Object Detection, Semantic Segmentation, or Scene Flow human annotations at the scale required by these perception systems are labor-intensive and complicated to obtain. This makes annotated data not only very expensive, but also prone to human errors due to the time-consuming and tedious nature of the process of manual data annotation. Also, with each new [LiDAR](#) sensor model, the annotation process has to be repeated.

Another important aspect of developing [AD](#) systems is the testing and validation of these systems. As the consequences of errors in [AD](#) systems can be potentially life-threatening, test coverage of these systems needs to be as comprehensive as possible.

However, real-world testing, recording and manual annotation for all scenarios that a vehicle might encounter is prohibitively expensive, especially considering the long tail of edge cases (i.e. seldomness of occurrence) that need to be covered. At the same time, the fast-paced AD development cycle requires quick iterations, which is difficult to achieve with real-world testing: Investigating the effect of changes to sensor hardware configurations etc. is impossible to do quickly and cheaply in the real world. Additionally, edge cases and other rare events can sometimes not be safely tested in the real world due to endangerment of humans or expensive prototypes.

In summary, the progress of AD technology is (among other reasons) hindered by the complexity and high cost of data annotation, and the cost of real-world testing, especially in seldom edge cases.

1.2 Contributions

In order to mitigate these challenges and to alleviate the burden of data annotation and real-world testing, this thesis investigates two different methods to reduce the cost of developing AD technology.

The first method investigated in this thesis is LiDAR simulation. In certain aspects, simulation has many advantages over real-world testing. Simulations allow for repeatable closed-loop simulation, which is useful for debugging and improving systems, as all (internal and external) parameters can be controlled precisely and independently. This control includes parameters that are impossible to control in the real world such as weather, traffic, even long-tail scenarios. However, simulating sensors like LiDAR, cameras, and RADAR with high fidelity is challenging and computationally expensive. Lacking fidelity or simulation artifacts can lead to systems overfitting to simulation, i.e. they perform well in simulation but fail to generalize to the more nuanced real world.

In this thesis, we develop a physically-based LiDAR simulator that can be used for training and evaluation of LiDAR object detection networks. Specifically for LiDAR simulation, the two most important aspects are the visibility aspect (“Does the sensor see the target?”), as well as the noise on the measurement (“How does the sensor see the target?”), which our simulator models and which we evaluate in this thesis. The visibility aspect is critical, since it dictates whether an object can be detected by a LiDAR sensor at all, which has severe impact on safety. Furthermore, real-world LiDAR sensors are subjected to noise, which is important to model in a simulator, as it directly influences the appearance of objects. Our simulator is designed to model these aspects as accurately as possible, based on the physical principles of LiDAR sensors. We compare and evaluate our simulator to existing simulation models in these aspects.

An alternative method to training networks on simulated data investigated in this thesis is self-supervised learning. Self-supervised learning can leverage large amounts of unlabeled data from the real world (i.e. the target domain), that systems can collect relatively inexpensively. This is an advantage over simulation, where discrepancies between the source domain (simulated data) and the target domain (real-world data) will cause performance degradation. However, self-supervised learning is challenging because of the

increased complexity compared to supervised learning: Self-supervised (proxy-) tasks need to be meaningful and beneficial to the actual optimization goal. The task, model capability and losses must be carefully designed and balanced, otherwise no meaningful representation can be learned by the network. In this thesis, we introduce a new state-of-the-art method for self-supervised **LiDAR** scene flow estimation and motion segmentation. Two distinct phenomena create observable scene flow from a sensor’s perspective: The first is the sensor’s ego-motion, the second is moving objects in the scene. Existing methods only recognize this fact, but we leverage it to create a self-supervised training signal for motion segmentation. This in turn improves the estimated scene flow, demonstrating the mutual benefit of simultaneous motion segmentation and scene flow estimation.

Finally, we use the motion cues contained in the **LiDAR** scene flow to develop a framework for self-supervised **LiDAR** object detection. We achieve this by introducing a novel self-supervised self-training framework: The self-supervised 3D scene flow is used to generate initial pseudo ground truth for the object detection task, and with the help of our novel self-supervised trajectory-regularized self-training framework, we are able to train state-of-the-art single-frame 3D object detection networks without using any human annotations. In extensive experiments, we demonstrate that our self-supervised framework outperforms other self-supervised methods. Furthermore, despite the fact that the self-supervised **LiDAR** scene flow is used for bootstrapping the pseudo ground truth, we show that the final object detector generalizes to all relevant movable objects.

1.3 Thesis Outline

Chapter 2 gives an introduction over the technical fundamentals required to follow this thesis, conveying a basic understanding for the topics touched. Then, Chapter 3 reviews the related work in the field of **LiDAR** simulation and self-supervised learning in the context of **LiDAR** scene flow and object detection. Chapter 4 presents the development of a physically-based **LiDAR** simulation model for training and evaluation of **LiDAR** object detection networks. Chapter 5 introduces our novel method for self-supervised **LiDAR** scene flow estimation and motion segmentation. The insights from this chapter are used to create a framework for self-supervised **LiDAR** object detection in the following Chapter 6. The thesis is concluded in Chapter 7, including a discussion and suggestions for future work.

2 Background

This chapter provides an overview of the background knowledge required to understand the work presented in this thesis.

2.1 Time-of-Flight: Working Principle of LiDAR

Compared to other light-based sensors such as cameras, [Light Detection and Ranging \(LiDAR\)](#) sensors have a completely different working principle. Cameras use passive, light sensitive sensors, i.e. they measure light emitted by an external light source (such as the sun or a flash), that is reflected from a target and then focused by lenses onto a light-sensitive sensor plane. The exposure of a pixel is continuous w.r.t. the power of the light that arrives at the sensor, i.e. the darker the scenery, the more underexposed the pixel will be. The stronger an object reflects the surrounding light into the camera, the brighter the resulting pixels will be in the image. For individual pixel values, color and brightness of the object is measured, but no direct distance information is available.

[LiDAR](#) sensors on the other hand are active sensors, providing their own light source via [Light Amplification by Stimulated Emission of Radiation \(laser\)](#) technology. A [laser](#) is a device that emits coherent light, i.e. light with a single wavelength (monochromatic), and a single phase. Energy is introduced into the [laser](#) material to “pump” up electrons to a higher energy level, a process referred to as electrical pumping. When the excited electrons fall back down to their normal energy level, they release electromagnetic waves with a single wavelength and synchronized phase, i.e. [laser](#) light. For further information on lasers, see the excellent foundational book by Vogel et al. [[Vog12](#)].

To conduct a measurement, a [laser](#) pulse is fired towards a potential target. Provided the target is close enough and reflective enough, the [LiDAR](#) receptor can detect the returning/reflected [laser](#) pulse by means of a photo diode. The *time of flight* Δt of the [laser](#) pulse is recorded and the distance of the sensor to the target (range R) computed using the speed of light c , see Eq. (2.1) and Fig. 2.1.

$$R = \frac{\Delta t c}{2} \quad (2.1)$$

This sensor technology is thus usually referred to as time-of-flight sensing.

In addition to the range R , the strength of the reflected [laser](#) pulse is also measured, most often referred to as intensity. The strength and shape of the echoing pulse is heavily influenced by material properties of the target, as well as angle of incidence of the incoming/returning [laser](#) pulse. Generally speaking, the darker the color of a target, the less reflected energy there will be in the returned pulse. When the reflected [laser](#) pulse is too

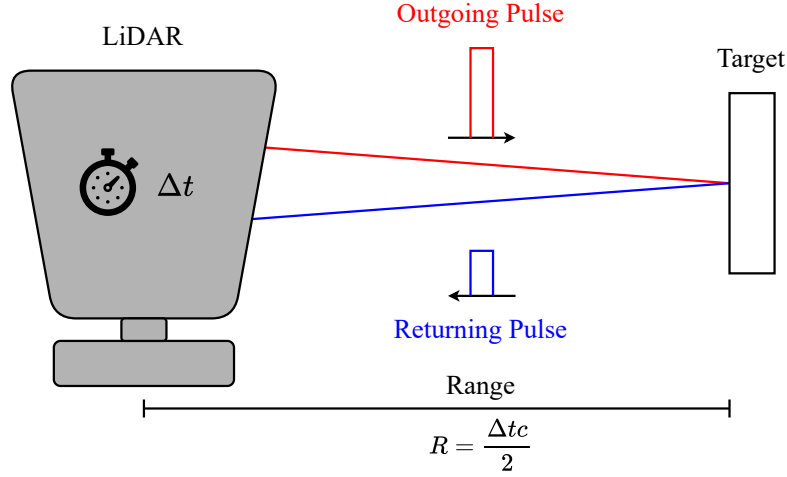


Figure 2.1: Time-of-flight measurement principle of **LiDAR** sensors: The travel time Δt of the **laser** pulse is measured, and the range R to the target is calculated using the speed of light c .

weak to be detected by the receiver, the sensor returns range $R = \infty$, i.e. the measurement is unsuccessful and thus invalid. For further information of target geometry and color on the measurement, see [BM18] or Section 4.2. This measurement process is repeated for many vertically stacked lasers, many times per second, usually while the sensor is rotating at 10 Hz to 20 Hz around its up-axis, resulting in a constant stream of reflected points being picked up by the sensor. While there exist **LiDAR** sensors which move the **laser** around by other means than rotation (i.e. flash **LiDAR**, or microelectromechanical systems (MEMS) **LiDAR** with snake pattern), they are not as common as the rotating ones used in the most popular **AD** datasets, such as the KITTI dataset [GLU12b], nuScenes [Cae+20], WOD [Sun+20] or AV2 [Wil+21] datasets, which are used in this work.

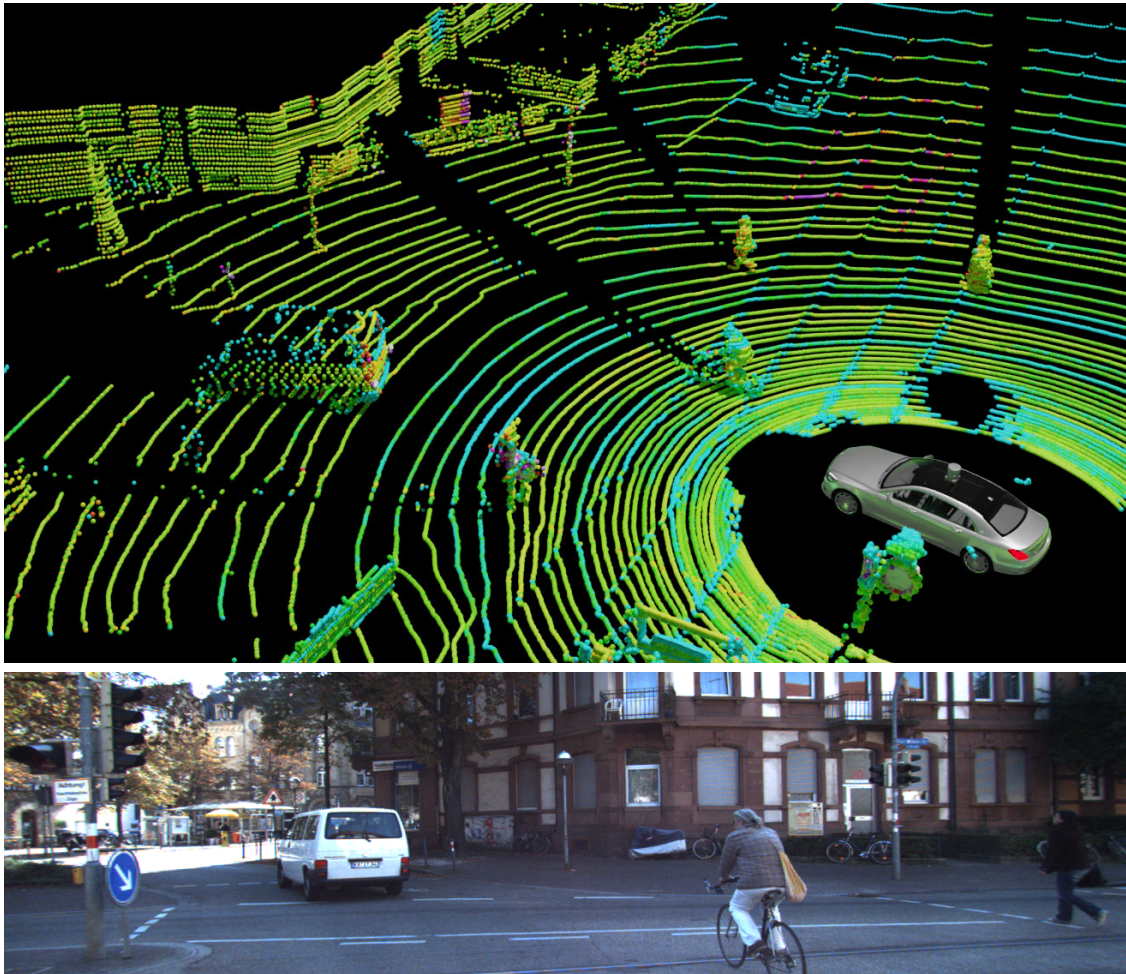
The measured range R , in combination with the elevation angle θ (which can be obtained through the intrinsic calibration parameters of the sensor) and azimuth angle ϕ (the measured rotation angle of the sensor), can be converted to Cartesian coordinates, allowing the **laser** to create a sparse 3D representation of its surroundings, with a much wider **Field Of View (FOV)** than regular cameras. One 360° sweep of accumulated points is often referred to as **point cloud**, cf. Fig. 2.2.

2.2 Point Clouds and Range Images

A point cloud is a set of points in 3D space: $\mathcal{P} = \{\mathbf{p}_i, i \in \{0, \dots, N-1\}\}$. Each point consists of coordinates $\mathbf{p}_i = (x, y, z)^T$. Sometimes, a point cloud is also described in a matrix form, where each row represents a point, and the columns represent the coordinates of the points: $\mathcal{P} \in \mathbb{R}^{N \times 3}$, $\mathbf{p}_i \in \mathbb{R}^3, i \in \{0, \dots, N-1\}$.

An alternative representation of a point cloud is the so-called **range image**, where the range R is stored in a 2D grid from the perspective of the sensor, i.e. the measured range

values are stored in a grid, with the location in the grid being determined by the azimuth and elevation angles of each measurement, cf. Fig. 2.3.



(a) A point cloud (colored by intensity) with its corresponding camera image from the KITTI dataset.



(b) Color mapping for normalized scalar values, used for intensity above.

Figure 2.2: Points are colored by their intensity (colormap at the bottom). Blue is low intensity, red is high intensity, with green and yellow in between.

However, as the sensor is usually moving with the ego-vehicle and recording a continuous stream of points due to its rotation, the origin of the measurement coordinate system is shifting over time, resulting in a slightly motion-distorted point cloud. For static elements (w.r.t. the world), this effect can be corrected using ego-motion information, which can be estimated using [Simultaneous Localization and Mapping \(SLAM\)](#) algorithms, [LiDAR odometry](#) or [Global Positioning System \(GPS\)/Inertial Measurement Unit \(IMU\)](#) data.

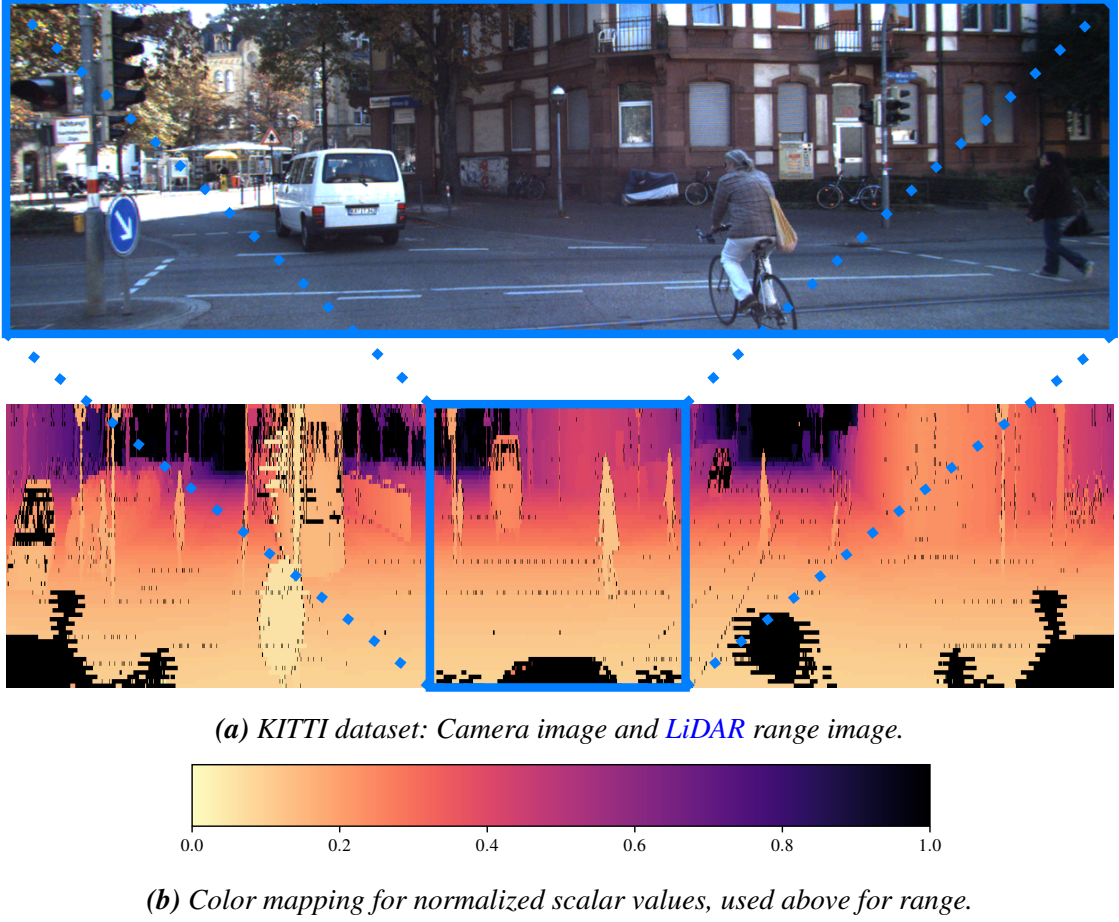


Figure 2.3: Black pixels in the range image indicate missing measurements. The range image is generated from the *LiDAR* point cloud using [Tri+20], clipped at 50 m and normalized, while vertical resolution is increased eightfold for better visualization. The *FOV* of the forward-facing camera is marked in blue in the two domains for better visual orientation. Note the difference in *FOV* between the camera ($\approx 90^\circ$) and the *LiDAR* sensor 360° .

2.3 Lidar Scene Flow

Scene Flow refers to the three-dimensional "vector/displacement field" of points between two consecutive point clouds. It is closely related to optical flow, which deals with two-dimensional pixel displacement between two consecutive images.

Given two consecutive point clouds $\mathcal{P}_t \in \mathbb{R}^{N \times 3}$, $\mathbf{p}_i \in \mathbb{R}^3, i \in \{0, \dots, N-1\}$ and $\mathcal{P}_{t+1} \in \mathbb{R}^{M \times 3}$, scene flow represents the 3D displacement vector $\mathbf{f}_i$ for each point i in the first point cloud $\mathcal{P}_t$ to its corresponding location in the second point cloud $\mathcal{P}_{t+1}$ w.r.t. the observing sensor frame. Note that there are no one-to-one correspondences between the points in the two point clouds (i.e. $\mathbf{p}_i + \mathbf{f}_i \notin \mathcal{P}_{t+1}$) since both point clouds are sparse, unordered and in most cases contain different numbers of points, see also Fig. 2.4. This makes the scene

flow estimation problem challenging. The full scene flow for two point clouds going from one point cloud $\mathcal{P}_t$ to the next $\mathcal{P}_{t+1}$ is represented as a set of 3D vectors $\mathcal{F} \in \mathbb{R}^{N \times 3}$, where $\mathbf{f}_i \in \mathbb{R}^3, i \in \{0, \dots, N-1\}$. $\mathcal{F}$ contains valuable information about the movement of points and the sensor and can thus help to understand the dynamics of a scene, as can be seen in Fig. 2.5.

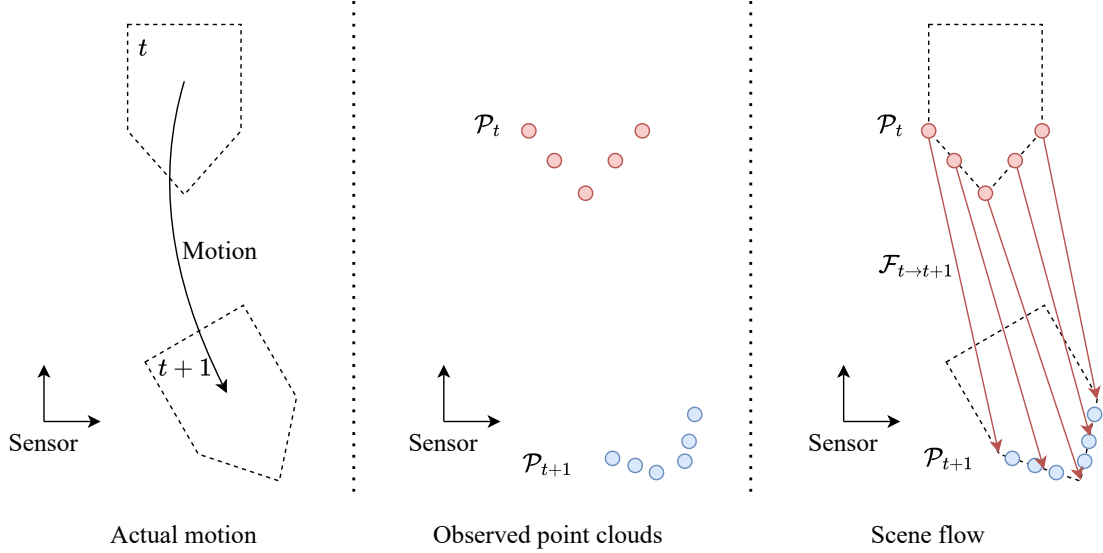


Figure 2.4: 3D Scene Flow: A sensor records two consecutive point clouds $\mathcal{P}_t$ and $\mathcal{P}_{t+1}$. Between the recording of the two point clouds, targets in the scene move relative to the sensor due to their own motion or the sensors ego-motion or a combination thereof. Since the recorded point clouds are sparse, unordered and do not have one-to-one correspondences over time, the goal of 3D scene flow estimation is to predict the 3D displacement vector $\mathbf{f}$ for each point in the first point cloud $\mathcal{P}_t$ to its corresponding location in the second point cloud $\mathcal{P}_{t+1}$: $\mathcal{F}_{t \rightarrow t+1} = \{\mathbf{f}_i \in \mathbb{R}^3, i \in \{0, \dots, N-1\}\}$. This means, that there might not be a measured point at the estimated scene flow target location $\tilde{\mathbf{p}}_i = \mathbf{p}_i + \mathbf{f}_i \notin \mathcal{P}_{t+1}$.

2.3.1 Kabsch Algorithm

The Kabsch algorithm [Kab76] can be used to obtain the optimal similarity transformation that brings two *corresponding* lists of points as close together as possible. Given two point clouds $\mathcal{P} \in \mathbb{R}^{N \times 3}$ and $\mathcal{Q} \in \mathbb{R}^{N \times 3}$ with corresponding points $\mathbf{p}_i$ and $\mathbf{q}_i$ each, it minimizes the mean squared error

$$e^2(\mathbf{R}, \mathbf{t}, s) = \sum_i \|\mathbf{q}_i - s\mathbf{R}\mathbf{p}_i - \mathbf{t}\|^2 \quad (2.2)$$

to find the optimal rotation matrix $\mathbf{R}$, translation vector $\mathbf{t}$ and scaling factor s . In the context of perfectly estimated scene flow, two consecutively recorded point clouds $\mathcal{P}_t$ and



Figure 2.5: A point cloud with its corresponding *LiDAR* scene flow from the KITTI dataset. Ego vehicle is standing still, and the *LiDAR* scene flow is dominated by the forward motion of the cars (traffic is flowing towards the right in the image). The points are colored by the magnitude and orientation of the scene flow vector given the color wheel in the upper left corner.

$\mathcal{P}_{t+1}$ do not have correspondences. On top of this, points in the first point cloud shifted by their scene flow are not present in the second point cloud, i.e. $\mathbf{p}_i + \mathbf{f}_i \notin \mathcal{P}_{t+1}$. However, we can create an artificial consecutive point cloud $\mathcal{Q}$ with known correspondences, by simply shifting the first point cloud $\mathcal{P}_t$ by the estimated scene flow $\mathcal{F}$ towards the second point cloud, i.e. $\mathbf{q}_i = \mathbf{p}_i + \mathbf{f}_i$, circumventing the correspondence problem. For *LiDAR* point clouds, we can further assume scaling factor $s = 1$, and apply the Kabsch algorithm to find the optimal rigid body transformation $\mathbf{T}$ that minimizes the root mean squared error between two point clouds:

$$\mathbf{T} = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{pmatrix}, \mathbf{T} \in \mathbb{R}^{4 \times 4}, \mathbf{T} \in \mathbf{SE}(3) \quad (2.3)$$

Most scenes are dominated by static background objects, and only a few objects are moving. Scene flow of the points falling on these static objects is caused by the ego-motion of the sensor, i.e. the majority of points in the scene will have scene flow that corresponds to the rigid body motion of the sensor between the two point clouds:

$$\begin{pmatrix} \mathbf{f}_i \\ 0 \end{pmatrix} = \begin{pmatrix} \mathbf{q}_i \\ 1 \end{pmatrix} - \begin{pmatrix} \mathbf{p}_i \\ 1 \end{pmatrix} = \mathbf{T} \begin{pmatrix} \mathbf{p}_i \\ 1 \end{pmatrix} - \begin{pmatrix} \mathbf{p}_i \\ 1 \end{pmatrix} = (\mathbf{T} - \mathbf{I}_4) \begin{pmatrix} \mathbf{p}_i \\ 1 \end{pmatrix}. \quad (2.4)$$

Each corresponding pair of points can be weighted by a weight $w_i > 0$, $w = \sum_i w_i$. To compute the sensor ego-motion, the weight would be $w_i = 0$ for points belonging to

dynamic objects, and $w_i = 1$ for objects that belong to the static background. Here, $\hat{\mathbf{f}}_i$ represents the scene flow estimated by a network:

$$\hat{\mathbf{T}} = \operatorname{argmin}_{\mathbf{T} \in \mathbb{R}^{4 \times 4}} \sum_{\mathbf{p}_i \in \mathcal{P}_t} w_i \left\| (\mathbf{T} - \mathbf{I}_4) \begin{pmatrix} \mathbf{p}_i \\ 1 \end{pmatrix} - \begin{pmatrix} \hat{\mathbf{f}}_i \\ 0 \end{pmatrix} \right\|^2 \quad (2.5)$$

The solution for the translation is the mean flow

$$\hat{\mathbf{t}} = \frac{1}{w} \sum_i w_i \hat{\mathbf{f}}_i. \quad (2.6)$$

Inserting this into the original optimization goal (2.5) we get

$$\hat{\mathbf{R}} = \operatorname{argmin}_{\mathbf{R} \in \mathbb{R}^{3 \times 3}} \sum_{\mathbf{p}_i \in \mathcal{P}_t} w_i \left\| \mathbf{R} \mathbf{p}_i - (\mathbf{p}_i + \hat{\mathbf{f}}_i - \hat{\mathbf{t}}) \right\|^2. \quad (2.7)$$

The Kabsch algorithm solves this via the cross-covariance matrix $\mathbf{H} \in \mathbb{R}^{3 \times 3}$

$$\mathbf{H}^{(j,k)} = \frac{1}{w} \sum_i w_i \mathbf{p}_i^{(j)} \left(\mathbf{p}_i^{(k)} + \hat{\mathbf{f}}_i^{(k)} - \hat{\mathbf{t}}^{(k)} \right) \quad (2.8)$$

(written here in component notation). Then, singular value decomposition (SVD) is applied to the covariance matrix $\mathbf{H}$:

$$\mathbf{H} = \mathbf{U} \mathbf{\Sigma} \mathbf{V}^T, \quad (2.9)$$

where $\mathbf{U}$ and $\mathbf{V}$ are orthogonal, and $\mathbf{\Sigma}$ is diagonal. The rotation can then be calculated via

$$\hat{\mathbf{R}} = \mathbf{V} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & d \end{pmatrix} \mathbf{U}^T \quad (2.10)$$

using $d = \operatorname{sign}(\det(\mathbf{V} \mathbf{U}^T))$ as correction value.

The resulting transform $\hat{\mathbf{T}}$ is differentiable w.r.t. the weights and scene flow.

2.3.2 Metrics

Average Endpoint Error (AEE) is the most commonly used metric used to evaluate the performance of scene flow algorithms. It measures the average Euclidean distance (**Endpoint Error (EE)**) between estimated 3D motion vector $\mathbf{f}_i$ and the ground truth flow vector $\mathbf{f}_{\text{gt},i}$ for each point $\mathbf{p}_i$ in a point cloud $\mathcal{P}_t \in \mathbb{R}^{N \times 3}$. The ground truth flow vectors are either part of a dataset (see also Section 2.8) or can be obtained by combining the ego-motion of the sensor with the tracked motion of the objects in the scene.

$$\text{AEE} = \frac{1}{N} \sum_i \text{EE}_i = \frac{1}{N} \sum_i \left\| \mathbf{f}_i - \mathbf{f}_{i,\text{gt}} \right\|_2 \quad (2.11)$$

2 Background

Since most scenes in **AD** datasets are dominated by static background, **AEE** is often reported separately for static and moving objects, since correct scene flow estimation is more challenging and important for moving objects.

Similarly, the relative error (RE) is defined as the ratio of the **EE** and the ground truth motion vector length:

$$\text{RE}_i = \frac{\text{EE}_i}{\|\mathbf{f}_{i,\text{gt}}\|_2} = \frac{\|\mathbf{f}_i - \mathbf{f}_{i,\text{gt}}\|_2}{\|\mathbf{f}_{i,\text{gt}}\|_2} \quad (2.12)$$

Multiple inlier and outlier statistics based on RE and **EE** are established in the community of scene flow estimation [**LQG19**; **Gu+19**; **PBM20**; **Wan+20c**]:

Strict Accuracy (AccS) is the fraction of all points where $\text{EE} < 0.05 \text{ m}$ or $\text{RE} < 0.05$.

Robust Accuracy (AccR) is the fraction of all points where $\text{EE} < 0.1 \text{ m}$ or $\text{RE} < 0.1$.

Outliers (Outl) are the fraction of all points where $\text{EE} > 0.3 \text{ m}$ or $\text{RE} > 0.1$.

2.4 LiDAR 3D Object Detection

For a given **LiDAR** point cloud, the 3D object detection task refers to predicting 3D upright oriented boxes for traffic participants (vehicles, cyclists, pedestrians), see Fig. 2.6. In some datasets, traffic signs and traffic lights are also considered as objects.

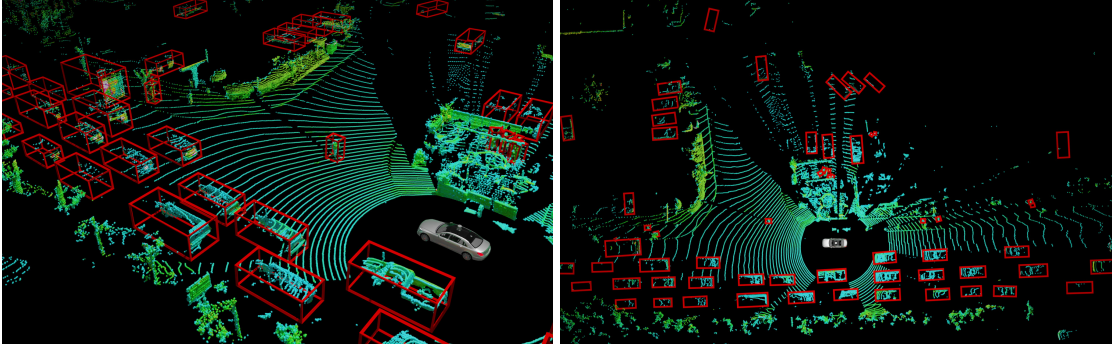


Figure 2.6: LiDAR Object Detection, same point cloud from two different angles, ground truth boxes in *red* for reference, points colored according to intensity.

A set of 3D bounding boxes $\mathcal{B}$ at time t is usually represented as:

$$\mathcal{B}^t = \{\mathbf{B}_j^t, j \in \mathbb{N}\} = \{(x, y, z, l, w, h, \theta, \mathbf{c})_j\} \quad (2.13)$$

Here, (x, y, z) refers to the center position, (l, w, h) to the length, width and height of the box, and θ to the heading (orientation around up axis) of the box. $\mathbf{c}$ is a vector representing the estimated (possibly unnormalized) class probability distribution for each box by the network, i.e. the estimated confidence values. For ground-truth boxes, the confidence vector $\mathbf{c}$ is a one-hot vector, i.e. only one class has a confidence of 1, all others have confidence 0.

To measure the performance of object detectors, the task is formulated as a classification problem and evaluated using classification metrics. The performance is then measured

over a validation/test split of the respective dataset. Firstly, predictions are thresholded by their predicted confidence, those with lower confidence than the *confidence threshold* are discarded. A detection is considered to be correct (i.e. true), when a certain matching criterion (2D [Intersection over Union \(IoU\)](#), 3D [IoU](#) or center distance) is better than a certain *matching threshold* for a predicted and a ground truth box. TP refers to the number of true positive detections, FP to the number of false positive detections (Type 1 Error), FN to the number of false negative detections (Type 2 Error).

Precision is the ratio of correct predictions by the model in relation to all predictions made by the model for a specific *confidence threshold* and *matching threshold*.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad (2.14)$$

Recall is the ratio of correct predictions by the model in relation to all objects present in the ground truth for a specific *confidence threshold* and *matching threshold*.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (2.15)$$

A single value for TPs, FPs, FNs and thus Precision and Recall always corresponds to a single *confidence* and *matching* threshold.

When the *matching* threshold is kept fixed, varying the *confidence* threshold can then be used to create the characteristic Precision-Recall curve. The main metric to measure object detection performance is [Average Precision \(AP\)](#), which equates to the area under the Precision-Recall curve.

$$\text{AP} = \int_0^1 \max\{p(r') | r' \geq r\} dr, \quad (2.16)$$

where $p(r)$ is the Precision-Recall curve over varying *confidence* thresholds. In practice, the metric samples a fixed number of score thresholds over the range of predicted *confidence* scores, and thus approximates the Precision-Recall curve. Usually, [AP](#) is stated in combination with the *matching* threshold used and the relevant class, like so: [AP](#)(@0.5IoU3D,Car) = 0.56. To arrive at a single metric to compare models capable of distinguishing different object classes (i.e. cars, pedestrians, ...) the [AP](#) score is often averaged over all N number of classes in the dataset and then referred to as [Mean Average Precision \(mAP\)](#)

$$\text{mAP} = \frac{1}{N} \sum_i^N \text{AP}_i. \quad (2.17)$$

2.5 Unsupervised Learning

Unsupervised learning targets learning from data without using human annotations, learning only from the inherent structure of the data. This is in contrast to supervised learning,

which requires availability of explicitly annotated (labeled) data. Unsupervised learning can be helpful to gather knowledge about unknown data which has no labels and to find inherent structure and patterns in the data. It includes many different techniques, such as clustering, dimensionality reduction, anomaly detection or generative modeling. **Clustering algorithms** like DBSCAN [Est+96] group data points based on proximity, revealing groups in the data. For instance, clustering can segment customers with similar purchasing behaviors in a retail database, while image segmentation methods can identify regions with similar pixel characteristics in images, which is for example useful for image analysis. **Dimensionality reduction** methods, such as Principal Component Analysis (PCA) [FRS01] reduce the number of dimensions of data points in a dataset while preserving its essential structure, for easier visualization and analysis. Another dimensionality reduction technique are autoencoders [HS06], which learn to efficiently encode and reconstruct data. On the one hand, the learned representations can be used for further machine learning tasks, such as classification. On the other hand, autoencoders can also be used for **Anomaly detection**, where a high reconstruction error (comparing the encoded and reconstructed data to the input) indicates unusual or previously unseen data points. **Generative Models** are designed to generate novel, but plausible data points by capturing the underlying data distribution of a training set. This can be for example achieved via an adversarial setup (cf. [Generative Adversarial Network \(GAN\)](#) [Goo+14]) or a learned denoising process (cf. [Diffusion Models](#) [HJA20]).

The distinction between unsupervised learning and self-supervised learning, which will be discussed in the next section, is often blurred. For further information on unsupervised learning, see [Car+20; Che+24].

2.6 Self-Supervised Learning

Self-supervised learning is a subfield in the area of unsupervised learning.

It is a method where the supervisory signal for a model and task are generated from the data itself, instead of relying on manual (human) annotations. To this end, inherent structures of data and the world can be leveraged, such as temporal consistency, e.g. by predicting the next frame in a video [SMS15; LKC17]. Characteristic relationships inside the data can also be used as supervision signal, like when training a network to colorize grayscale images or videos [CYS15; ZIE16]. In most cases, the goal of self-supervised learning is to force the model to learn meaningful representations of the data. By fine-tuning the pre-trained model on very little labeled data, the model can then be used for a variety of downstream tasks. Beyond the obvious benefit of cost reduction, self-supervised learning does not suffer from label corruption to the same extent as its supervised counterpart [Hen+19]. But self-supervised learning also comes with drawbacks: Generally speaking, self-supervised learning is much more complex than supervised learning, as it involves pretext and/or auxiliary tasks that require their own losses and hyperparameters, which need to be understood and tuned. Furthermore, it is computationally more expensive, as it requires more data and more training time as the supervisory signal is usually weaker than for supervised learning.

In many cases, self-supervision can serve as a form of pre-training. For example in natural language processing, a popular pre-training objective is to predict the next word in a text, given the previous words. This forces the model to learn about the inherent structure, grammar and relationships in language and between words and sentences without the need for any labeled data beyond sensible text, as the data serves as its own supervision [SVL14; Rad+18; Dev+19]. The pre-trained model can then be fine-tuned and used for a variety of downstream tasks such as summarizing texts or translating between languages to only name a few.

2.7 Self Training

Self training is a sub-discipline of semi-supervised learning which can be used when only a small fraction of the dataset has (manual) annotations available. In the beginning (sometimes referred to as bootstrapping phase), a model is trained on the small fraction of the dataset for which annotations are available. This model is then utilized to generate pseudo ground truth for part of the unlabeled data, by thresholding the models' predictions with a certain confidence threshold. Alternatively, a fixed proportion of the most confident predictions is used as new pseudo ground truth. These pseudo ground truth examples are then added to the training data, and a new classifier is trained on the updated training set. This process is repeated until the model converges. The challenge with self training is to minimize the influence of label noise on the training. Naturally, classifiers make mistakes, and these mistakes are propagated as pseudo ground truth to the next iteration of the training, which degrades model performance. This challenge can be partially addressed by self-ensembling, i.e. keeping running averages of predicted pseudo ground truth over the entire training dataset at different stages of the training, and then filtering labels based on the agreement of predictions and pseudo ground truth from the training set [Ngu+20]. In summary, self training is a highly data-efficient method to obtain models with usable performance. By iteratively improving the training set, models significantly outperform those trained solely on the small annotated part of the dataset. For further information on self training there exists an excellent survey on self training by Amini et al. [Ami+22].

2.8 Datasets for AD

In this work, we use multiple datasets, for which we give a brief overview and comparison in the following and Table 2.1. We focus only on datasets that contain LiDAR data, are (at least partly) annotated with 3D bounding boxes, and have precise vehicle ego-motion (through fused GPS and IMU) available. Track IDs for the 3D bounding boxes in combination with ego-motion allows us to generate ground truth LiDAR scene flow, which we need to evaluate our methods in Chapter 5.

2 Background

	KITTI[GLU12b]	nuScenes[Cae+20]	WOD[Sun+20]	AV2[Wil+21]
Scenes	156	1000	1150	1000
Annotated Lidar Frames	15k	40k	230k	300k
Hours	1.5	5.5	6.4	8.3
3D Boxes	80k	1.4M	12M	10M*
Lidars	1	1	5	2
Cameras	4	6	5	9
Avg. Points/point cloud	120k	34k	177k	107k
Maps	No	Yes	No	Yes
Cities	1	2	3	6

*Table 2.1: Comparison of popular AD datasets. *: Excluding test set.*

2.8.1 KITTI

The KITTI dataset [GLU12b] is one of the earliest datasets for AD research. It contains a variety of sensor data, such as LiDAR, multiple cameras, GPS and IMU data. The dataset consists of multiple categories, such as object detection, object tracking, odometry and stereo scene flow. All data has been recorded in and around the city of Karlsruhe, Germany. The LiDAR data is recorded using a Velodyne LiDAR sensor type “HDL64” (HDL64) sensor, which is a 64-layer LiDAR sensor. While some parts of the dataset have been annotated by humans (in the forward-facing camera FOV), a large part of the dataset has not been annotated and is therefore ideally suited for self-supervised learning.

2.8.2 nuScenes

The nuScenes dataset [Cae+20] has been recorded more recently in 2018, containing a large amount of sensor data, such as LiDAR, camera, GPS and IMU data. The dataset has been recorded in Boston and Singapore, and features a large amount of annotated data (object bounding boxes have been annotated at 2 Hz), as well as basic map data of the environment. The LiDAR data is recorded using a Velodyne VLP32C LiDAR sensor with 32 layers, rotating at 20 Hz. It features full 360° coverage in terms of cameras. In total, 1000 scenes have been recorded, each spanning around 20 seconds.

2.8.3 Waymo Open Dataset(WOD)

The WOD dataset [Sun+20] is a large-scale dataset for LiDAR object detection, motion prediction, 3D semantic segmentation, and occupancy and flow prediction. It is recorded in multiple cities in the US. It has been recorded using five proprietary LiDAR sensors and consists of 1150 scenes spanning around 20 seconds each. One long range LiDAR sensor is mounted on the roof of the vehicle with a 360° FOV, and four additional short range sensors have been mounted at the corners of the vehicle. The range of the LiDAR data has been restricted to 75 m, but in contrast to the other datasets, the first two returns

of each [laser](#) pulse are provided. Beyond reporting the mere coordinates and intensity of the [LiDAR](#) measurements, they also provide elongation for each [LiDAR](#) measurement, i.e. the elongation of the [laser](#) pulse relative to its nominal duration.

2.8.4 Argoverse 2(AV2)

The AV2 dataset [[Wil+21](#)] is a large-scale dataset for [AD](#) research. It has been recorded in multiple cities in the US, using two stacked Velodyne VLP32C [LiDAR](#) sensors with 64 layers combined, rotating at 20 Hz each, synchronized to point in opposite directions at all times. In addition to that, seven ring cameras, and two stereo cameras, along with high precision [GPS](#) and [IMU](#) data have been recorded. It consists of 1,000 thirty-second sequences.

3 Related Work

As this thesis touches on the topics of **LiDAR** simulation, self-supervised scene flow, and self-supervised **LiDAR** object detection, we provide an overview of related work in these fields. We begin by discussing the current state of **LiDAR** simulation in Section 3.1. In Section 3.2, we discuss the state-of-the-art in scene flow and motion segmentation, specifically in the context of self-supervised learning. Finally, in Section 3.3, we give an overview over the state-of-the-art self-supervised **LiDAR** object detection.

3.1 **LiDAR** Simulation

A good general overview of the overall state of simulation (and by extent synthetic datasets) in the automotive domain can be found in [Kau+21; Son+24]. In the image domain, an abundance of research has been conducted in order to synthesize photorealistic images using Computer Graphics and **DL** [Su+15; Dos+15a; PJH16; Ros+16; Hof+18; Lee+19]. Not as much research has been conducted into the simulation of realistic **LiDAR** point clouds, which we discuss in the following.

3.1.1 Conventional Model-based **LiDAR** Simulation

Currently, there exist mainly two types of **LiDAR** simulators: The first is aerial **LiDAR** simulators, which are aimed at simulating airborne **LiDAR** systems used to map large areas for geo-scientific purposes, and the second is **AD LiDAR** simulators, which are aimed at simulating **AD** scenarios. The main difference between these two types of simulators is the scale of the simulated environment and more importantly the technology used in these **LiDAR** systems. Generally speaking, aerial **LiDAR** systems are more advanced and have a much larger range than automotive **LiDAR** systems, as they are not as limited in terms of size and power consumption. Furthermore, in contrast to automotive **LiDAR**, aerial **LiDAR** data is often analyzed only post-flight, allowing the recording and usage of higher-fidelity signals (full waveform).

Aerial **LiDAR Simulation** Aerial **LiDAR** simulators such as [BH16; Kim19] are aimed at simulating airborne **LiDAR** systems for geo-science applications. While these **LiDAR** models are useful for finding flight parameters for geo-scientific mapping experiments, their applicability for autonomous driving is limited. The modeled large scale aerial **LiDAR** systems use technologies that are not available in automotive **LiDAR** sensors, such as full waveform (i.e. continuous detailed) recording of the returned **LiDAR** pulse for example, which can be many meters in diameter.

AD LiDAR Simulation is aimed at simulating AD scenarios for the purpose of training and validating AD systems. While many of them provide assets, scenarios and multi-agent simulation support, many of these do not have realistic LiDAR models [KH04; Sha+17; Lab19]. They provide the user with unrealistic (i.e. perfect and unbounded) range values or “only” purely random failed measurements. This contrasts heavily with real LiDAR systems, as the maximum detection range for a LiDAR system is finite and limited by the laser, material properties of the target, angles of incidence of the laser beam and weather effects. Unbounded measurement range leads to unrealistic detection performances, giving these simulators limited use for either validation and testing or data collection for training purposes due to the domain gap between simulation and reality.

Other frameworks such as Microsoft Airsim [Sha+17], or LGSVL [Lab19] are more focused on image-based methods, but provide many different assets like scenarios and different actors in their simulation frameworks. However, their LiDAR models neither model rolling shutter, which is characteristic for many LiDAR sensors, nor do they possess any measurement noise model for either intensities of returns, missing returns or measured ranges. The rolling shutter effect (i.e. movement of the sensor and its measurement during a sweep) introduces distortion and latency into the measurement procedure. Taking this latency into account is crucial at high speeds, as both the ego vehicle and vehicles coming towards the ego vehicle can move a significant distance during the measurement.

Hurl et al. [HCW19] introduced a LiDAR simulator integrated into the popular video game GTA V. However, its field of view is limited to 45° , it also does not possess a rolling shutter feature, and it possesses unrealistic infinite range. Both [Sun+22] and [Xu+22] are large-scale multi-modal synthetic datasets providing cameras, LiDAR, GPS, IMU and RADAR data. However, they both are static datasets, with fixed sensor models, which do not allow for the simulation of different sensor models or different sensor parameters. Furthermore, they ignore specific measurement- and noise characteristics of the extracted LiDAR point clouds and focus on the comparison of fusion strategies for AD.

Gschwandtner et al. [Gsc+11] were the first to introduce a LiDAR simulation model called **Blender Sensor Simulation Toolbox (BlenSor)** that models the noise affecting the range measurement. Of all available models, it is the most realistic LiDAR simulation in terms of physics: Besides Gaussian range noise, it also incorporates rolling shutter effect, plus it models the visibility of objects based on range and target color, see Fig. 3.1. However, their visibility model is not based on radiometry, but on a simple visibility model based on the range to the target and the reflectivity r of the target extrapolated from the two data points provided in the operating manual of the sensor: $d_{\text{lower}} = 50\text{m}$, $r_{\text{lower}} = 0.1$ for pavement and $d_{\text{upper}} = 120\text{m}$, $r_{\text{upper}} = 0.8$ for cars and foliage (here, d represents range to target and r the target reflectivity), cf. the sensor datasheet [Vel13] and Fig. 3.1.

Its range noise model assumes a Gaussian distribution for the range noise, ignoring the influence of the actual range to the target and other target properties. For example, Craig et al. [GL10] discovered a strong dependence of range errors on the angle of incidence. We improve over the mere assumption of applying Gaussian noise to measurements by deriving a LiDAR model based on radiometry. Also, while BlenSor only simulates noise on the measured range to an object, we simulate noise also in the equally important visibility aspect. We argue that overlooking objects due to missing detections is at least as

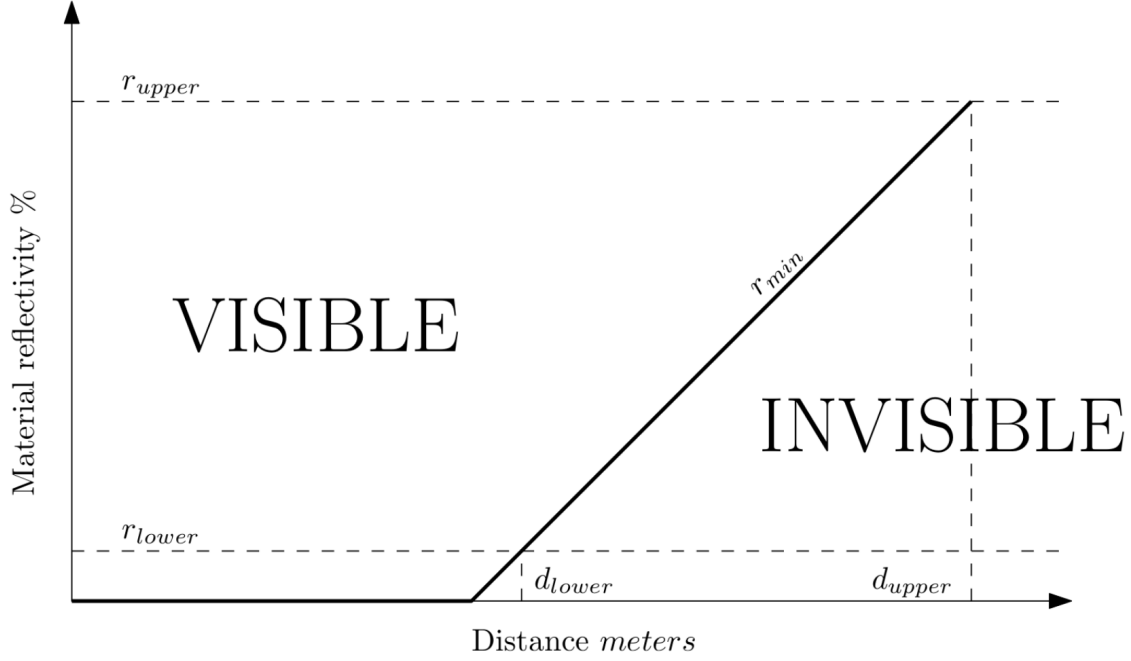


Figure 3.1: Image extracted from [Gsc13]. Visibility model as used by *Blensor*, based on the two data points provided in the operating manual of the sensor: $d_{lower} = 50$ m, $r_{lower} = 0.1$ and $d_{upper} = 120$ m, $r_{upper} = 0.8$, see [Vel13]. In practice, material reflectivity is assumed to be equal to the averaged RGB values of the target in the simulator.

important as realistic measurement noise properties.

3.1.2 Sim2Real Domain Adaption & Data-driven, Generative Simulation

More recently, Sallab et al. [Sal+19] suggest using *GAN* to perform style transfer of rasterized top view images of point clouds between the KITTI dataset [GLU12a] and the CARLA simulator. However, they only apply style transfer on *Bird's Eye View (BEV)* images between the domains, and **not** of point clouds, making them unusable for *LiDAR* simulation.

More recently, Nakashima et al. [NK21; NIK23] proposed a *GAN*-based approach to simulate *LiDAR* point clouds based on range images. Unfortunately, before the *GAN* is trained, the range images are downsampled to 25% of their original horizontal resolution, which is a significant reduction of the point clouds and limits the applicability of the model to real world scenarios. Furthermore, it is not possible to control the contents of the generated range images in any way, limiting the methods' applicability for simulation.

Guillard et al. [Gui+22] only simulate raydrops and intensity based on camera images, but not the actual range measurement. Using a *Convolutional Neural Network (CNN)*, they learn a mapping from RGB images to *LiDAR* intensity and raydrops on real-world data. Then, the learned model is used to generate *LiDAR* intensity and raydrops from

synthetic RGB images. It is unclear what the influence of the domain gap from real RGB images to simulated RGB images is on the predicted intensities and ray drops.

Another approach is [Man+20], who sample from a large database of accumulated multi-frame object point clouds and environments previously recorded with LiDAR. This means that their approach does not generalize to unseen objects or arbitrary environments, as the simulation capability is limited to previously recorded data.

Very recently, Zhang et al. [Zha+24] proposed to use a Neural Radiance Field (NeRF)-approach to re-simulate AD scenarios with realistic LiDAR point clouds. However, their approach is not generalizable to new scenarios, as each new scenario requires a new training of the NeRF model. This makes their approach computationally very expensive. Tonderski et al. [Ton+24] address the problem of long training times through a more efficient sampling strategy and a more streamlined network architecture compared to prior work, but assume completely rigid objects, not allowing full control over actor states. Furthermore, while the NeRF-based approaches produce realistic-looking point clouds, they are not interpretable, as the underlying NeRF model is essentially a black box. Like [Man+20], NeRF-based approaches do not generalize to previously unseen objects or arbitrary environments for which no prerecorded data is available.

3.2 Scene Flow and Motion Segmentation

This section has been previously published in S. A. Baur, D. J. Emmerichs, F. Moosmann, P. Pinggera, B. Ommer, and A. Geiger. “SLIM: Self-Supervised LiDAR Scene Flow and Motion Segmentation”. In: *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*. IEEE, 2021, pp. 13106–13116. DOI: [10.1109/ICCV48922.2021.01288](https://doi.org/10.1109/ICCV48922.2021.01288). URL: <https://doi.org/10.1109/ICCV48922.2021.01288> [Bau+21].

The term *scene flow* was first introduced by Vedula et al. [Ved+99]. Traditionally, it is estimated using stereo cameras [CK02; PKF05; HD07; Wed+08; MHG15; VSR15; Jai+15; NFS15], and has successfully been applied to other domains such as RGB-D sensors [Sha+18].

3.2.1 Motion Segmentation

While there exists a group of methods layering optical flow into consistent groups, they cannot be easily transferred to point clouds as they make heavy use of color information and brightness constancy assumptions [SB12; WB14; SSP15]. Research for scene flow and motion segmentation on range images started with hand-crafted features and matching, see [MF10]. In this work, however, we focus on scene flow estimation in the point cloud domain, which provides its own challenges such as sparse data as well as the absence of well-established representations. Within recent years, the focus shifted towards learning-based methods, as discussed next.

3.2.2 Supervised Point-Cloud-Based Scene Flow

For many years, research in the area of point-cloud-based scene flow estimation focused on supervised learning methods. Most approaches relied on synthetic datasets [Gu+19; LQG19; LYB19; Wan+20c; Wei+20; PBM20; Wan+20a] or non-public datasets [Wan+18a] for training. Behl et al. [Beh+19a] started to use the real-world KITTI Tracking dataset, but it is only densely annotated in the camera FOV. Gojcic et al. [Goj+21] lessened the requirement on annotated data by using weak supervision through odometry and foreground-background segmentation to train their network for simultaneous motion segmentation and scene flow estimation, allowing them to use the SemanticKITTI [Beh+19b] dataset to train their approach.

3.2.3 Self-Supervised Learning

The dependence on costly annotated data can be overcome by self-supervised learning. It has been successfully applied to train networks on images to predict optical flow, stereo flow, depth (or a combination thereof) [BMK13; WG15; YS18; ZLH18; LTC19; CSS19; God+19; Ran+19; WJE19; Luo+20; HR20], and motion segmentation [Tun+17; BML18] using geometric and temporal/cyclic consistency losses. Self-supervised learning on point clouds, however, is a young field where in most cases self-supervision has also been used to pre-train a network on a proxy-task and to reuse or fine-tune the weights for the actual task using supervision [HH19; SS19; Xie+20; Pou+20; TAG20; Zha+21]. For a more general and broader introduction to self-supervised learning, we refer the interested reader to the excellent survey by Balestrieri et al. [Bal+23].

Image-based Self-Supervised Learning

In Computer Vision, FlowNet [Dos+15b] paved the way for deep learning based optical flow estimation. However, it required a large amount of labeled data for training, and dense optical flow is a challenging and labor-intensive task to annotate for humans. Instead, FlowNet relied on a large synthetic dataset for training, leading to the usual challenges with the domain gap between training and test data. Based on FlowNet, Yu et al. [YHD16] proposed a method for unsupervised learning of optical flow based on photometric constancy assumption using only real-world consecutive camera images. Basically, they formulated a loss that penalizes the difference between the intensity of a pixel in one image and the intensity of the corresponding pixel in the next image, differentially warped by the predicted optical flow, in combination with a smoothness assumption of the predicted optical flow field, used as regularization. This method was able to achieve state-of-the-art results on the KITTI dataset [GLU12b]. This marked the beginning of a new era in optical flow estimation, where self-supervised learning became the new standard for training optical flow networks [Ren+17; Wan+18b; Jan+18; Liu+19].

Self-Supervised Scene Flow on Point Clouds

Recently, there has been some research into self-supervised scene flow estimation on point clouds, but all of these works have various shortcomings.

Point Cloud Size Many architectures are based on graphs [Tis+20; PHL20], PointNet/FlowNet3D [Qi+17; MOH20] or layouts with coarse global all-to-all correlations combined with finer local correlations [KER20]. In order to cope with exploding memory requirements during training, they rely on random downsampling of their input point clouds, in many cases also during inference. Mittal et al. [MOH20] subsample to 2,048 points, others [Wu+20; KER20; PHL20; Tis+20] downsample to 8,192 points. These methods perform well on point cloud data gathered from stereo cameras or RGB-D data sources with little sparsity, such as KITTI-SF [MHG18] or FT3D [May+16]. In comparison, LiDAR sensors cover much larger areas than stereo rigs but at a much lower point density. Regular LiDAR sensors measure upwards of 100k 3D points in each frame. We are able to process much more points than previous works. This gives our architecture (presented in Chapter 5) an advantage over existing methods, as it can leverage arbitrary point densities during both training and inference.

Correspondence Assumption Another serious drawback in current methods (at the time of publication) was discovered by Zuanazzi et al. [Zua+20]: They note that [Wu+20] and [MOH20] train (and evaluate) on point clouds with a 1-to-1 correspondence $\mathbf{p}_i^{(t+1)} = \mathbf{p}_i^{(t)} + \mathbf{f}_i$, the KITTI-SF and FT3D dataset. Furthermore, they show in experiments heavily degrading performance when points are shuffled and sampled in an effort to destroy the correspondences prior to training/inference. Real-world measurements from LiDAR sensors do not abide by the correspondence assumption, as the scene is sampled for each time step, and a bijective mapping between point clouds does not exist.

Robustness to Outliers Since flow vectors on static points all derive from the same sensor motion, Tishchenko et al. [Tis+20] introduced a two-stage-network: The first stage, a pose regression network, estimates ego-motion and transforms the source point cloud. The second stage subsequently estimates flow. Although realistic data recordings contain much more static than moving points, which should lead to good performance, they suffer from high outlier rates on real data - even on static points. We demonstrate in our experiments (cf. Chapter 5) a much better outlier resistance predicting dense scene flow for all points by regressing the sensor’s ego-motion using the fully differentiable Kabsch algorithm [Kab76].

Self-Supervised Motion Segmentation on Point Clouds

To the best of our knowledge, at the time of publication there exists only one previous network that learns motion segmentation on point clouds using self-supervision: Thomas et al. [Tho+20] use multiple recordings of the same environment to train a network to identify permanently static, slow, and fast-moving objects. Our method (presented in

Chapter 5) does not require multiple recording passes and can train on pairs of point clouds directly.

3.3 LiDAR Object Detection

LiDAR object detection refers to the task of detecting objects in one or multiple consecutive point clouds. For a brief introduction into LiDAR object detection see Section 2.4.

This section has been previously published in S. A. Baur, F. Moosmann, and A. Geiger. “LISO: Lidar-Only Self-supervised 3D Object Detection”. In: *Computer Vision - ECCV 2024 - 18th European Conference, Milan, Italy, September 29-October 4, 2024, Proceedings, Part LXXXVI*. Ed. by A. Leonardis, E. Ricci, S. Roth, O. Russakovsky, T. Sattler, and G. Varol. Vol. 15144. Lecture Notes in Computer Science. Springer, 2024, pp. 253–270. DOI: [10.1007/978-3-031-73016-0_15](https://doi.org/10.1007/978-3-031-73016-0_15). URL: https://doi.org/10.1007/978-3-031-73016-0_15 [BMG24].

3.3.1 Single Frame LiDAR 3D Object Detection

Object detectors operating on 3D point clouds are an active research field. The currently best-performing ones are using deep neural networks trained via supervised learning and can be categorized by their internal representation: Some networks operate on points directly like PointRCNN [SWL19], 3DSSD [Yan+20], and IA-SSD [Zha+22]. Others project the points either to a virtual range image [Mey+19; Sun+21; Lia+21] or into a voxel representation like VoxelNet [ZT18], PointPillars, CenterPoint [YZK21], and Transfusion-L [Bai+22]. However, all aforementioned methods require large human-annotated datasets in order to perform well and obtaining such annotations is very expensive. We address this problem in this thesis, enabling training of state-of-the-art object detectors using pseudo ground truth.

3.3.2 Object Distilling from Motion Cues

Multiple approaches have been suggested which leverage motion cues from LiDAR frame pairs in order to detect *moving* objects. Using the assumption of local geometric constancy, they decompose dynamic scenes into separate moving entities by applying as-rigid-as-possible optimization. Examples of these are the works by Dewan et al. [Dew+16], RSF [DZ23] and OGC [SY22]. The two former methods are optimization-based whereas OGC uses these constraints as loss-function to self-supervise a segmentation network. Although these methods avoid the need for expensive labels, in contrast to our method they tend not to work well in low-resolution areas, they are typically slow, and can only detect *moving* but not *movable* (i.e. static but potentially moving) objects.

3.3.3 Pseudo Ground Truth for Object Detection

Different approaches have been proposed to mine pseudo ground truth for training object detectors: Najibi et al. [Naj+22] and Seidenschwarz et al. [Sei+24] use motion cues similar to Section 3.3.2 in order to distill *moving* objects as pseudo ground truth and use it to train an object detector. [Naj+22] runs optimization for each frame pair to obtain LiDAR scene flow, clusters points, fits boxes, tracks them using a Kalman Filter and finally refines them on ICP-registered point clouds. [Sei+24] optimizes a clustering algorithm on motion cues through message passing which they then apply to segment point clouds into a set of instances and subsequently fit boxes. As shown in [Sei+24], and in contrast to our approach, both methods suffer from a large performance gap between *moving* and *movable* objects. We demonstrate that our method does not suffer from this gap.

[Har+21], [Shi+22] were the first to *iteratively* apply a detection, tracking, retraining paradigm to AD data. They build upon consistency constraints between object detectors which they train in the LiDAR and camera domain, making use of an optical flow network which is trained using supervision. The approach requires a calibrated sensor rig with possibly large camera coverage, IMU, and precision GPS. Similarly, [WCZ22] uses video sequences together with LiDAR scene flow to jointly train a camera and a LiDAR object detector. MODEST [You+22; XW23] does not require coverage by calibrated cameras, but instead adds the additional requirement to having multiple LiDAR recordings of the same location in order to identify objects that vanished over time. With such demanding requirements, these approaches are not easy to use and are partially unsuited for popular AD datasets.

Oyster [Zha+23] is the approach most similar to ours. It uses DBSCAN [Est+96] on LiDAR point clouds to initialize pseudo ground truth. Clusters are then tracked using forward-and-reverse tracking in sensor coordinates (i.e. without consideration for ego motion), using a complex policy for confidence-based track retention. After training an object detector on close range data, they employ zero-shot generalization to the far range data, track again and iteratively retrain the detector. In their experiments they use different hyperparameters for different datasets. Our method, in contrast, explicitly considers sensor motion, produces much cleaner initial proposals by leveraging a self-supervised LiDAR scene flow network, does not require zero-shot-generalization from near-to-far-field, and works with current state-of-the-art object detectors. We show on multiple datasets that our method outperforms [Zha+23], that it works robustly with the same set of hyperparameters and also generalizes well to detect *movable* objects.

4 Physically-based LiDAR Simulation

This chapter deals with the design of a physically-based LiDAR simulation model with realistic noise patterns that closely mimic those of real-world sensors. In Section 4.1, we begin by motivating what makes physically-based LiDAR simulation desirable, especially compared to conventional, heuristics- or DL-based simulation methods. This is then followed by a deeper introduction into the working principle of LiDAR sensors in Section 4.2. In the following, Section 4.3 then introduces and derives our physically-based LiDAR simulation model. We then discuss parameter selection for our model in Section 4.4 and provide implementation details in Section 4.5. Beyond the visual comparison in Section 4.6.2, we evaluate our approach in three key aspects: First by comparing statistical properties of a real-world sensor with our simulation model in Section 4.6.3. Secondly, we evaluate suitability of our LiDAR simulation as data-generation engine for training of various LiDAR object detection models in Section 4.6.4. Thirdly, we evaluate its suitability to validating real-world object detectors in staged simulation scenarios in Section 4.6.5. The chapter is then closed by a conclusion and discussion in Section 4.7.

4.1 Introduction

The widespread realization of AD faces many challenges, one of them being that improvements in accuracy and performance of current supervised learning approaches scale unfavorably w.r.t. the required size and variety of annotated datasets. Another challenge is safety and safety-guarantees from a regulatory standpoint. It is impossible to record and annotate data for any imaginable or unimaginable scenario that might occur in the real world to validate AD methods for various reasons: Firstly, the costs involved are prohibitive, and in some extreme cases, the process can be dangerous. Additionally, rare or “long-tail” incidents occur so infrequently that capturing them is exceedingly unlikely. Simulation can help to address these problems.

There are several benefits to using simulation compared to doing real-world testing. The most relevant being the cost-effectiveness: Instead of having to build costly hardware setups, parameters for the simulation can be varied cheaply, and their influence on the system performance can be evaluated quickly: For example, parameters that would take months or even years to change in a LiDAR sensor, such as number of lasers, rotation frequency, laser power or FOV can be changed in seconds in simulation. In addition to the full control over the internal model parameters, simulation also enables testing in a controlled environment, i.e. all external parameters can be adjusted fully and independently, allowing to simulate situations that would be too dangerous in real life or put expensive experimental vehicles at risk. Additionally, simulation facilitates easy

and automatic closed loop testing of changes applied to an AD stack, such as changes in the perception, sensor fusion or behavior system, without having to do real-world testing. This enables faster testing and development cycles, as simulators can be run in parallel for multiple configurations compared to expensive hardware setups, which are not trivial to duplicate. Last but not least, since the internal state of the simulator is known at all times, it is possible to generate large amounts of labeled ground truth data, which can be used to train DL models or to validate existing systems extensively on rare events. This can help to ensure safe operation for rare events where a sufficient amount of data cannot possibly be collected in the real world.

General Challenges of Simulation While simulation has many advantages over real-world testing and data collection, it also brings with it its own challenges. Taking algorithms developed in simulation and deploying them in the real world requires a high degree of confidence in the simulation. Therefore, the simulation must be as realistic as possible, and its limitations must be well understood and tested. While simulating traffic scenarios, including agent behavior and lane graphs on an abstract level (i.e. without sensor data and perception algorithms) is reasonably well understood [FV10; Lop+18], simulating realistic sensor data is challenging. For camera data it is possible to leverage photo-realistic video game engines to simulate very realistic-looking camera data, provided the scene is well modeled. However, simulating RADAR or LiDAR remains a challenge, since their working principle differs significantly from cameras, and thus these modalities are not able to profit from the advancements in photo-realistic rendering made over the last decades. At the same time, the quality (i.e. realism) of simulated sensor data is crucial to its usefulness, as any difference between the real and simulated sensor data will lead to different behavior in the other domain. Simulated sensors must exhibit the same characteristics as real sensors, i.e. have the same limitations (artifacts, measurement errors, ...).

Challenges of LiDAR Simulation A sophisticated and realistic LiDAR simulation model needs to approximate two important LiDAR sensor characteristics: Firstly, the visibility aspect of targets needs to be addressed, i.e. what the sensor can detect and what it cannot detect based on range, sensor- and target properties. Secondly, a concept about different sources of noise during the measurement process, as well as their influence on the measured time-of-flight and ultimately the resulting noisy measured range $\hat{R}$.

However, most existing simulators rely primarily on geometry to determine what is visible and what is not, providing users with a “perfect” view of the world that lacks realism. These simulators often ignore important factors such as the varying physical interactions of different materials with light, sensor technology, and their limitations. This means that the simulated sensor can see objects at infinite range and with infinite measurement precision, which is not the case for real sensors. There are multiple ways to make a simulation more realistic: One way is to add noise to the final measurements, similar to BlenSor [Gsc+11]. Another way is to use sim2real methods, which try to make simulated data look more like real data by employing domain adaptation methods

[Hof+18; Lee+19]. There is also the possibility to use completely data driven simulation (inserting measurements from real objects into simulated scenes) like Manivasagam et al. [Man+20] as described in Section 3.1, in combination with a deep-learning based predictor to predict visibility of targets, limiting the use of the simulator for validation purposes due to missing interpretability. Existing generative approaches such as [Sal+19; NK21; NIK23] are based on GANs, that lack interpretability and are difficult to control to produce specific desired outputs. Additionally, they work on downsampled point clouds, which do not capture the full complexity of the environment.

Interpretability and the Advantages of Physics A LiDAR simulation model allows researchers and engineers to analyze the performance of their perception algorithms in different scenarios. Being able to pinpoint certain behavior of the perception system under test to specific physical properties of the simulation model is a key advantage of a physically-based simulation model. This enables targeted optimization of the sensor model and the overall perception system. For example, the visibility of an object can be traced back to the material properties of the object and the angle of incidence of the laser beam or the laser power. This is not possible with data-driven simulation models, as the visibility of an object or measurement errors are not directly traceable to the input parameters or mathematical properties of the model, but are a result of the training data being condensed into black-box models.

4.2 Working principle of LiDAR

LiDAR sensors are time-of-flight sensors as introduced in Section 2.1. For a well calibrated LiDAR, reducing the “exposure” (i.e. laser power Φ_{P_E}) gradually will lead to no change in range measurements. When the returning signal becomes too weak to be detected no measurement can be obtained. There is no gradual degradation as there is with cameras: “Underexposed” laser targets result in no measurement, referred to as raydrop in the following.

This “**limited visibility**” aspect is crucial to modeling realistic measurement characteristics. In reality, the maximum safe speed of an AD vehicle is limited by the maximum range of the sensors, system performance and the overall system latency. Unfortunately, the maximum range at which an object can still be measured is strongly dependent on the material characteristics of the object [BM18; MHB20], prohibiting a simple visibility model solely based on range. Thus, sensor datasheets always contain the maximum range for a given target reflectivity: While a matte black object can become invisible at ranges larger than 50 m (cf. [Vel13; Gmb21]), a bright white object or retro reflector can be visible up to 120 m or more, depending on the specific LiDAR model. Fig. 4.1 shows how the returning signal of a LiDAR pulse can vary greatly depending on the material properties and orientation of the target.

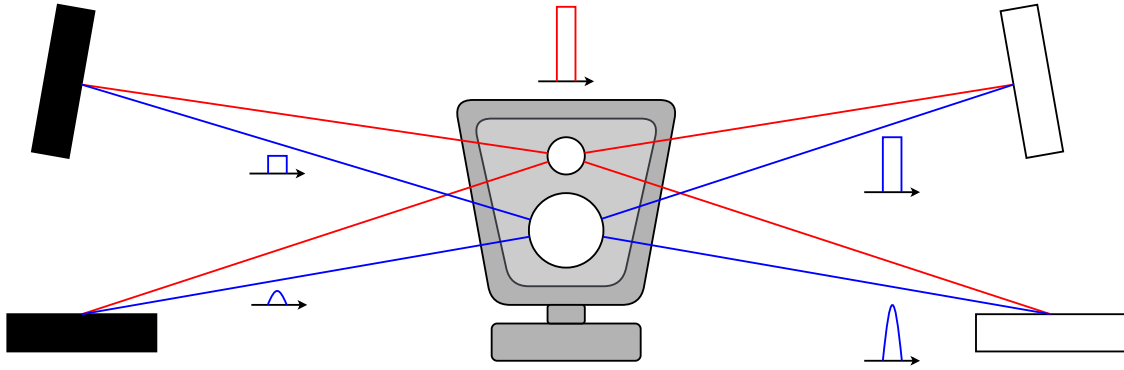


Figure 4.1: The sensor (shown in gray) emits four identical *laser* pulses (*red*). Depending on the material properties (shown in black/white) and orientation of the targets, the echo of the pulse (*blue*) varies greatly, despite equal range from the sensor to the targets. Whether a target can be picked up by the sensor thus depends greatly on the material properties and geometry of the target. For further information on the influence of target geometry and color on the *LiDAR* measurement process, see [BM18].

LiDAR Technical Components

LiDAR sensors consist of various optical and electronic systems, for a brief overview see Fig. 4.2. In terms of optics, there are lens systems for the *laser* emitting and *laser* receiving part of the sensor. The *FOV* of the emitter and receiver optics must overlap, defining the minimum distance for measurements. The most relevant systems and mechanisms are explained briefly in the following, more details can be found at [CC17; Ins18]. Conventional *LiDAR* sensors are made up of a fixed number of layers, each consisting of a matched pair of *Laser Emitter* (LE) and *Laser Receiver* (LR) with respective optical system, and measurement electronics. The power of each *laser* pulse varies slightly from shot to shot as well as between different layers cf. [GL10]. As each layer has one pair of LE and *Photo Diode* (PD) and thus can only measure one target at one point in time, most sensors rotate to scan the full environment, creating a constant stream of time-stamped range data.

Laser Emitter

The *Laser Emitter* (LE) is responsible for generating the *laser* pulse required for the measurement. Common *LiDAR* sensors either use 905 nm (e.g. HDL64) or 1400 nm (e.g. Luminar [ELW20]) wavelength. While the working principle is the same for both wavelengths, due to eye safety regulations 905 nm systems need to have lower power emitters. Higher wavelength *LiDAR* has drawbacks such as increased manufacturing cost (“fiber lasers”) and higher absorption in water droplets.

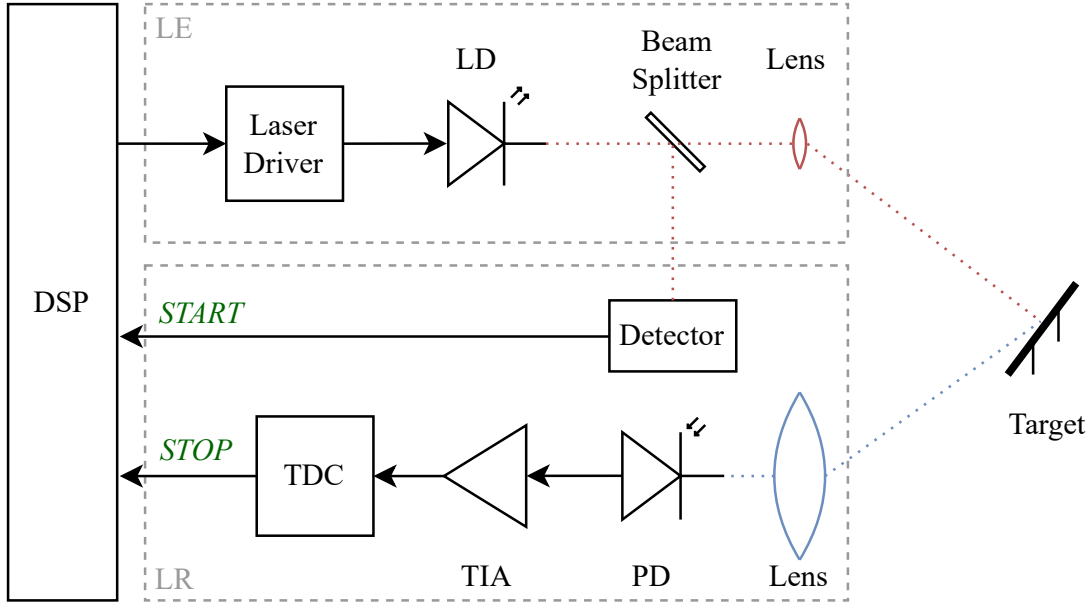


Figure 4.2: A basic LiDAR can be divided into two parts: The emitting part *Laser Emitter (LE)*, and the receiving part *Laser Receiver (LR)*. In order to conduct a range measurement, the *Digital Signal Processor (DSP)* triggers the *laser driver*, which in turn causes the *Laser Diode (LD)* to send out a *laser pulse*. The emitted pulse is split by a semipermeable mirror (sometimes called beam splitter), with a part of the pulse being sent to a detector, that triggers the start of the time measurement in the *DSP*. The other part of the pulse traverses the emitter lens, and is scattered at the target and picked up by the receiving optics: The photons are picked up by a *Photo Diode (PD)*, which outputs a current. This current is then converted to a voltage by a *Transimpedance Amplifier (TIA)* and amplified. The *Time Digital Converter (TDC)* then discretizes the signal by comparison to the visibility threshold, sometimes also referred to as dark threshold calibration value Φ_{P_x} . Finally, the *DSP* performs leading and falling edge detection, inferring range R and intensity of the return.

Photo Diode

The **PD** is responsible for picking up the signal reflected by the target and collected by the receiving optics. Similarly to a solar cell, the photosensitive semiconductor diode converts the incoming *laser* power P_R , i.e. returning photons, to an electrical current. The task of the **PD** is very challenging, since it needs to work reliably across a wide variety of echo intensities, cf. Table 4.1. This means that it needs to be sensitive enough to react to low-power echoes, but at the same time must not be “blinded” or even destroyed by highly reflective (retro-reflective) targets. For more details on (avalanche) photo diodes, see [Wil17; CC17].

Range [m]	Power [μ W]
10	80
30	9
100	0.8

Table 4.1: Returning pulse power at the receiver for identical target at different ranges. (Table extracted from [CC17]). Note how the power decreases by more than two orders of magnitude over the range to the target.

Transimpedance Amplifier

The TIA converts the current from the PD to a voltage signal, cf. [CC17]. The TIA is required to amplify the signal to a level that can be processed by the TDC in the next step.

Time Digital Converter

The TDC compares the signal to a threshold and discretizes the signal. The threshold Φ_P needs to be well calibrated: A threshold too low will result in so-called “ghosting”, as measurements are triggered by reflected sunlight or noise in the signal chain. Too high of a threshold, and the maximum range of the sensor will be reduced, as far away targets will not yield sufficiently strong returning pulses. Also, dark matte objects might become invisible to the sensor. As far away targets become indistinguishable against the background noise, the threshold is always a compromise between noise suppression and long range visibility. For more details on the inner workings of the TDC, please see [Inc15].

Digital Signal Processor

The DSP measures the time it takes from the laser pulse being fired until there is a significant change in the output of the TDC (time-of-flight principle, cf. Section 2.1). The DSP then measures the duration (i.e. it counts the number of clock ticks) that the TDC output signal is active. Comparing the measured number of clock ticks to the expected number of ticks indicates how the laser pulse was spread over time.

For more detailed information on the working principle of LiDAR, please refer to [KK11; Wil17; KNK17; CC17; Ins18].

4.3 Physical Model

Based on the working principle of LiDAR introduced in Section 2.1 and Section 4.2, we derive a physically-based LiDAR simulation model.

Goal

The goal is to build a mathematical model, that, given a target with certain material properties and orientation Θ can predict the returning signal of a LiDAR sensor, and from that derive a realistic binary visibility signal $V \in \{0, 1\}$, and where applicable, a noisy range measurement $\hat{R}$. The model should be parameterized by LiDAR sensor parameters Φ , based on realistic, physical units.

Preface

Instead of modeling each electrical component and its effects on a per-component basis as discussed in Section 4.2, we aggregate certain components and effects into larger functional blocks. For example, the PD converts incoming laser power to current, which is then converted to voltage by the TIA. We omit the conversion of physical units, as it is a linear operation. Also, since the TIA can be considered linear, i.e. performing solely signal amplification and conversion from current to voltage, we can omit it from the model and pretend that the PD directly outputs a power signal, without neglecting any crucial effects.

Defining Characteristic of the Pulse Echo

The pulse echo is characterized by two key properties: The **strength** and **shape** of the returning pulse. The **strength** of the pulse is dictated by the ratio of reflected and incoming power returning from the target surface: This depends on geometry and material parameters Θ , and not on the LiDAR parameters Φ beyond the emitted laser power Φ_E . The **shape** of the returning pulse essentially refers to the temporal aspect, i.e. which part of the target surface is illuminated (and reflecting back to the sensor) at which point in time. This is sometimes summarized under the designation “elongation” of the returning pulse. It only depends on geometry and the LiDAR parameters Φ , *not* directly on material parameters Θ .

4.3.1 Model Overview

This independence of the different aspects allows us to separate our simulator into isolated components: One that deals with the static component and is only dependent on the target material and geometric properties and one that deals with the shape of the reflected pulse (temporal aspect). The following subsections each correspond to a component in Fig. 4.3.

1. **Time-Surface Model:** This component models the amount of reflecting surface at the target over time (i.e. elongation) of the returning laser pulse, see Section 4.3.2.
2. **Surface-Reflection Model:** This component models the ratio of incoming and returning laser power reflected back into the sensor based on the material properties and orientation of the target surface. These properties and the process are constant for a measurement, see Section 4.3.3.

4 Physically-based *LiDAR* Simulation

3. **Threshold-Intersection Model:** This mechanism determines whether a point is visible to the sensor or not, by comparing the returning pulse power to the sensors' threshold Φ_P , see Section 4.3.4.
4. **Noise Model:** This introduces noise into the modeled measurement process, see Section 4.3.5.

These models are combined to form the complete *LiDAR* simulation model, cf. Fig. 4.3. The following sections describe the individual components in more detail.

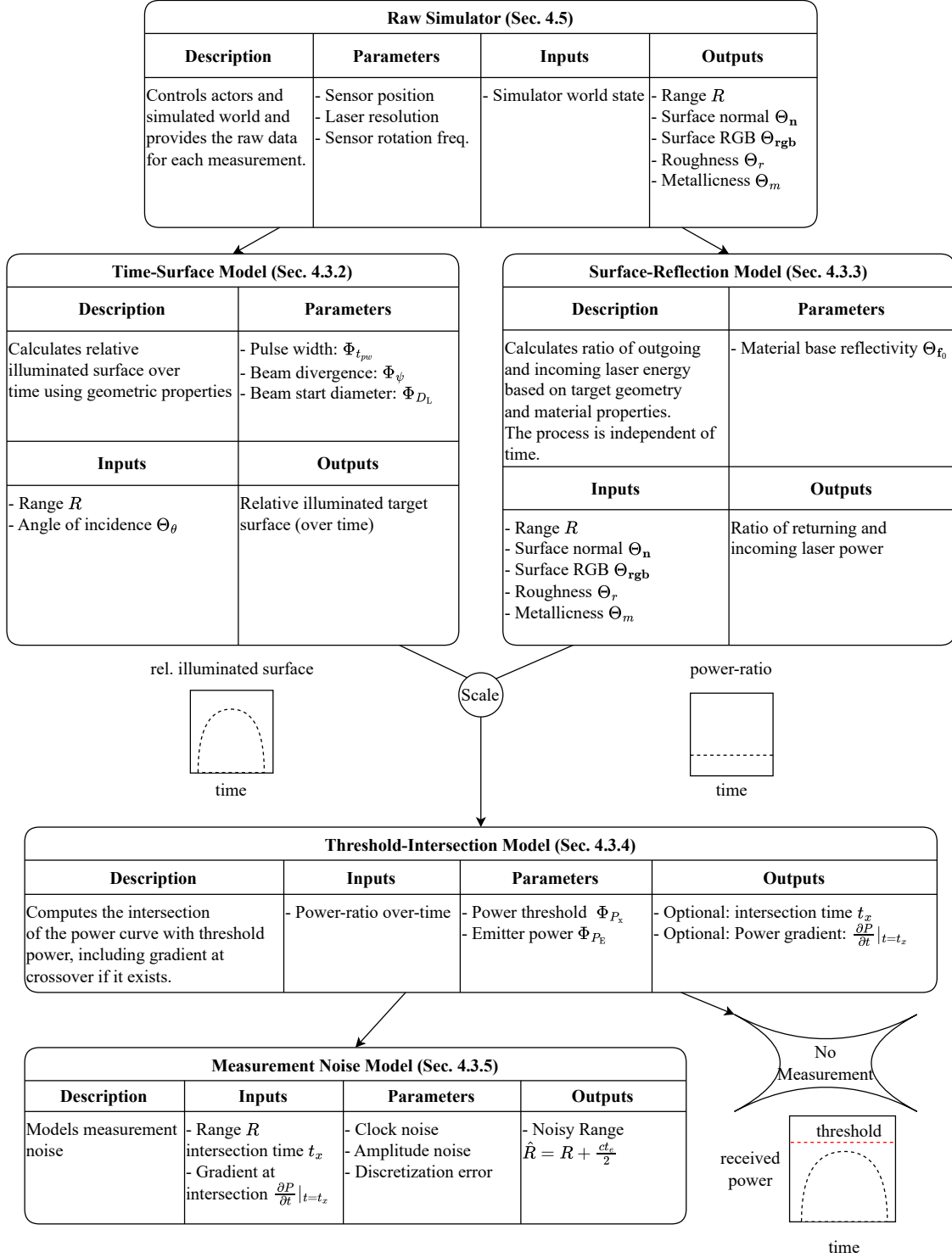


Figure 4.3: Overview over the whole simulation pipeline. Each component corresponds to a section.

4.3.2 Time-Surface Model

This model describes what part of the target area is illuminated at what point in time, i.e. what ratio of the total target surface is reflecting light back at the sensor.

The backscattering cross-section σ describes the ratio of incident electromagnetic energy on a target and the amount of energy scattered back into the direction of the incident electromagnetic wave. This has been extensively studied in the context of *RADAR* systems [Sko01], with entire books being dedicated to the topic of *RADAR* cross-sections [KST04], because this property essentially dictates the visibility of aircraft to *RADAR* systems.

The radar range equation (cf. [Sko01] and (4.1)) describes the power received P_r by a radar system as a function of the transmitted power P_t , the gain of the transmitting antenna G_t , the area of the receiving antenna A_r , the backscattering cross-section of the target σ , and the range R to the target:

$$P_r = \frac{P_t G_t A_r \sigma}{(4\pi)^2 R^4} \quad (4.1)$$

That is for the static case, with co-located transmitter and receiver systems. Wagner et al. [Wag10] adapted this concept from *RADAR* to *LiDAR*, as the working principle is quite similar (not to be confused with "Doppler *RADAR*", which is used to measure velocity). Additionally, we introduce the temporal aspect, modeling the change of illuminated area over time. For our purposes, the emitted *laser* pulse is modeled as a rectangular function with pulse width Φ_{tpw} . The beam divergence Φ_ψ causes the beam width w to increase with the range R to the target. The exit diameter of the *laser* emitter (LE) Φ_{DL} at the sensor in combination with the beam divergence Φ_ψ dictates the beam width w :

$$w(R) = \Phi_{DL} + 2R \tan \frac{\Phi_\psi}{2}. \quad (4.2)$$

The area perpendicular to the *laser* ray at the target is called the *laser* footprint A_{lf} :

$$A_{lf} = \frac{\pi w(R)^2}{4}. \quad (4.3)$$

Ω_s is the solid angle of the receptor, i.e. the area of the receptor optics over the range squared:

$$\Omega_s = \frac{\pi \Phi_{DR}^2}{4R^2}. \quad (4.4)$$

The backscattering cross-section σ is a product of the target area, the reflectivity of the target and Ω_s , describing the reflection characteristics (details see [RPB12], Eq. (14) from [Wag10]). The total illuminated area at the target must take the angle of incidence Θ_θ into account

$$\sigma = 4\pi \bar{f}_r A_{lf} \cos \Theta_\theta, \quad (4.5)$$

where $\bar{f}_r$ denotes the averaged *Bidirectional Reflectance Distribution Function* (BRDF) over Ω_i (solid angle of the incident light) and Ω_s (radiant light), being closely related

to biconical reflectance. According to [Wag10], σ can also be formulated in terms of incident and reflected power:

$$\sigma = \frac{4\pi}{\Omega_s} \frac{P_R}{\Phi_{P_E}} A_{\text{lf}}. \quad (4.6)$$

Wagner et al. [Wag10] adapt the radar equation (4.1) for use with laser ranging systems

$$P_R = \frac{\Omega_s}{4\pi A_{\text{lf}}} \Phi_{P_E} \sigma, \quad (4.7)$$

or when taking atmospheric damping τ into account:

$$P_R = \frac{\Omega_s}{4\pi A_{\text{lf}}} \Phi_{P_E} \sigma \tau(2R). \quad (4.8)$$

Aerosols and dust dampen the laser signal as it travels through air, but we neglect atmospheric damping ($\beta \stackrel{!}{=} 0$) for our simulation as this term is rather insignificant compared to $\frac{1}{R^2}$ in (4.12), especially for clear weather conditions, where $\beta \approx 0.096 \text{ km}^{-1}$ [Woj+14], i.e. the meteorological visibility is over 20 km. Consequently, we omit $\tau(2R)$ and simplify (4.8):

$$\begin{aligned} \tau(2R) &= e^{-R\beta} = 1 \\ P_R &= \frac{\Omega_s}{4\pi A_{\text{lf}}} \Phi_{P_E} \sigma. \end{aligned} \quad (4.9)$$

From [Wag10] (Eq. 14), we know that we can express the backscattering cross-section with the help of a BRDF term $\bar{f}_r$:

$$\sigma = 4\pi \bar{f}_r A_{\text{lf}} \cos \Theta_\theta. \quad (4.10)$$

$\bar{f}_r$ essentially scales the amount of reflected power depending on the material parameters, please see Section 4.3.3 for details.

The illuminated surface varies over time, and thus we introduce $\tilde{A}(t)$ describing the relative illuminated surface over time. We get:

$$\sigma(t) = 4\pi \bar{f}_r A_{\text{lf}} \cos \Theta_\theta \tilde{A}(t). \quad (4.11)$$

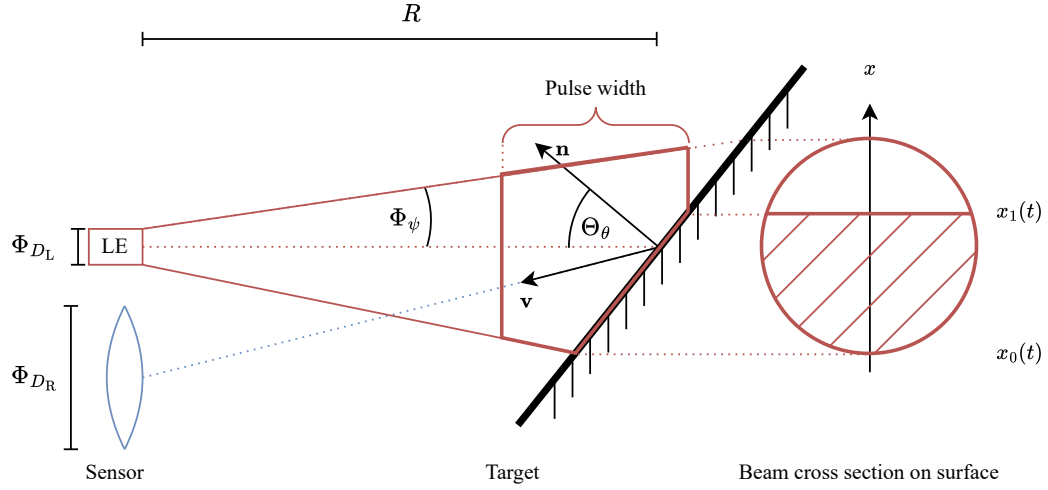
The LiDAR receiver in essence measures the returning laser power P_R over time:

$$P_R(t) = \frac{\pi \Phi_{D_R}^2}{4R^2} \Phi_{P_E} \bar{f}_r \cos \Theta_\theta \tilde{A}(t). \quad (4.12)$$

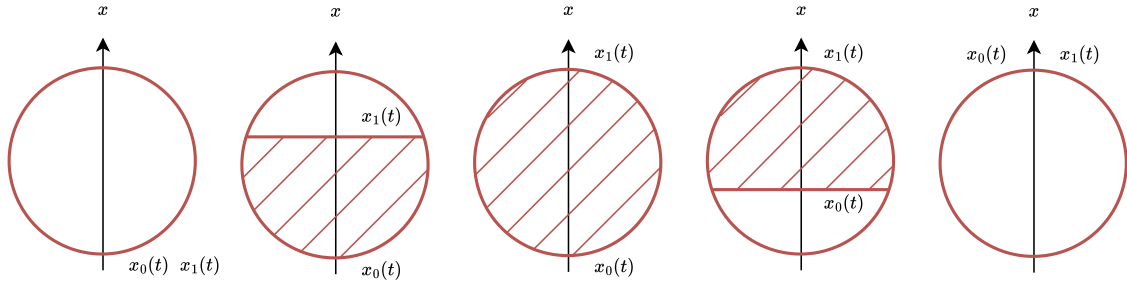
Relative illuminated surface over time: We define x as a coordinate across the laser footprint A_{lf} at the target, see Fig. 4.4. The range R across the target can now be expressed as a function of x :

$$R(x) = R + \Delta R = R + x \tan \Theta_\theta. \quad (4.13)$$

4 Physically-based LiDAR Simulation



(a) We introduce the helper coordinate x to describe the bounds of the illuminated surface at any given time t , depicted as hatched area in the figure.



(b) The illuminated target surface changes over time t . $x_0(t)$ and $x_1(t)$ represent its lower and upper bounds at time t .

Figure 4.4: Change of the illuminated target surface over time.

We assume the power emitted by the laser Φ_{PE} to be constant over time for a pulse. The amount and the location of target surface that is illuminated changes over time: $\tilde{A}$ is without unit, between 0 and 1 and refers to the relative amount of surface.

For simplicity, we formulate the response of the pulse to be symmetrical around $t = 0$ and t_0 and t_1 are $-\frac{\Phi_{tpw}}{2}$ or $\frac{\Phi_{tpw}}{2}$ respectively, marking the beginning and the end of the pulse relative to the center of the pulse.

We combine two Heaviside functions H to create the signal S , which indicates whether the surface is illuminated at time t

$$S(t) = H(t - t_0)H(t_1 - t) = H(t - 2\frac{R(x)}{c} - t_0)H(-t + 2\frac{R(x)}{c} + t_1), \quad (4.14)$$

and combine it with the size of the illuminated area at time t :

$$\tilde{A}(t) = \frac{1}{A_{\text{If}}} \int_{-\frac{w}{2}}^{\frac{w}{2}} 2\sqrt{\frac{w^2}{4} - x^2} S(t) dx. \quad (4.15)$$

Solving for zero crossings in the Heaviside function we get

$$\begin{aligned} t - 2\frac{R(x)}{c} - t_0 &= 0 \\ -t + 2\frac{R(x)}{c} + t_1 &= 0, \end{aligned} \quad (4.16)$$

and since

$$R(x) = R + x \tan \Theta_\theta, \quad (4.17)$$

we can solve for x and get

$$\begin{aligned} x_0(t) &= [(t - t_0)\frac{c}{2} - R] \frac{1}{\tan \Theta_\theta} \\ x_1(t) &= [(t - t_1)\frac{c}{2} - R] \frac{1}{\tan \Theta_\theta}, \end{aligned} \quad (4.18)$$

where $x_0(t)$ is the lower boundary coordinate of the illuminated surface, and $x_1(t)$ represents the upper bound for the illuminated surface at time t , see also Fig. 4.4b. Additionally, $x_0(t)$ and $x_1(t)$ are bounded by the beam width, giving us the integration boundaries u and l :

$$\begin{aligned} l &= \max(x_0(t), -\frac{w}{2}) \\ u &= \min(x_1(t), \frac{w}{2}). \end{aligned} \quad (4.19)$$

This allows us to rewrite (4.15) as

$$\tilde{A}(t) = \frac{1}{A_{\text{If}}} \int_l^u 2\sqrt{\frac{w^2}{4} - x^2} dx. \quad (4.20)$$

Finally, integration leads to:

$$\tilde{A}(t) = \frac{1}{A_{\text{If}}} \left(u\sqrt{\frac{w^2}{4} - u^2} - l\sqrt{\frac{w^2}{4} - l^2} + \frac{w^2}{4} \left[\tan^{-1}\left(\frac{u}{\frac{w^2}{4} - u^2}\right) - \tan^{-1}\left(\frac{l}{\frac{w^2}{4} - l^2}\right) \right] \right). \quad (4.21)$$

4.3.3 Surface-Reflection Model

In order to make the simulation as realistic as possible, we also need to capture the influence of different target materials on the simulated measurements, as already alluded to by f_r in (4.12). However, up to this point, our model only takes geometric- and temporal effects into account. To address this, we introduce a BRDF into our simulation model, i.e. a function describing how light from a source is reflected off a surface. For a given

incident light direction ω_i and outgoing direction ω_o , a BRDF f_r essentially provides the ratio of reflected radiance and incident irradiance. This is required by (4.12), in order to scale the strength of the returning laser pulse according to the target material properties. We closely follow the methodology and equations proposed in [Kar13], since it is the BRDF used in the underlying game engine [Epi19] of the simulator, i.e. the material parameters in the CARLA simulator [Dos+17a] have been optimized for realistic visual appearance for this BRDF. In the following, we describe the key equations from [Kar13] and their application to our method, without going into extensive detail, as we do not significantly deviate from [Kar13]. For further reference, the interested reader is referred to the original paper [Kar13] or the excellent book on physically-based ray tracing by Pharr et al. [PJH16].

To model the amount of laser power that arrives back at the sensor, both diffuse $\mathbf{f}_{\text{lambert}}$ and specular reflection $f_{\text{cook-torrance}}$ are combined according to a weight k_d :

$$\mathbf{f}_r = k_d \mathbf{f}_{\text{lambert}} + f_{\text{cook-torrance}}. \quad (4.22)$$

The weight k_d is determined by the amount of specular reflection F , which in turn depends on the view- and half vector ($\mathbf{v}$ and $\mathbf{h}$), as well as the base reflectivity $\Theta_{\mathbf{f}_0}$:

$$k_d = 1 - F(\mathbf{h}, \mathbf{v}, \Theta_{\mathbf{f}_0}). \quad (4.23)$$

F is an approximation of the Fresnel equation, called Fresnel-Schlick approximation [Sch94]. It describes the reflection (and transmission) of light at the interface between two media, depending on the angle of incidence. $\Theta_{\mathbf{f}_0}$ represents the base reflectivity for non-metallic materials, i.e. the amount of specular reflectance for a light ray that is perpendicular to the target surface according to the Fresnel model. Since the base reflectivity is not given directly in the simulator, we use the approximation proposed by Karis et al. [Kar13], linearly interpolating between a fixed value and the base color according to the metallicness of the surface:

$$\begin{aligned} \Theta_{\mathbf{f}_0} &= (1 - \Theta_m)0.04 + \Theta_m \Theta_{\text{rgb}} \\ F &= F(\mathbf{h}, \mathbf{v}, \Theta_{\mathbf{f}_0}) = \Theta_{\mathbf{f}_0} + (1 - \Theta_{\mathbf{f}_0})(1 - (\mathbf{h} \cdot \mathbf{v}))^5. \end{aligned} \quad (4.24)$$

The diffuse reflection term can simply be modeled as:

$$\mathbf{f}_{\text{lambert}} = \frac{\Theta_{\text{rgb}}}{\pi}. \quad (4.25)$$

For the specular component, the Cook-Torrance model [CT82] is used, as it is also the basis for the BRDF and materials in Unreal Engine 4 [Kar13; Epi19], which is the underlying game engine used by CARLA [Dos+17a]. The Cook-Torrance Model is a microfacet model. It assumes that the surface is made up of many small individual facets (with varying surface normals), each of which reflect light according to the microfacet model, i.e. the aggregation of the behavior of these microfacet represents the observed scattering of a surface.

The Cook-Torrance BRDF is given as

$$f_{\text{cook-torrance}} = \frac{DFG}{4(\omega_o \cdot \Theta_n)(\omega_i \cdot \Theta_n)}, \quad (4.26)$$

which combined with (4.22) and (4.25) leads to

$$\mathbf{f}_r = k_d \frac{\Theta_{\text{rgb}}}{\pi} + \frac{DFG}{4(\omega_o \cdot \Theta_n)(\omega_i \cdot \Theta_n)}. \quad (4.27)$$

D is the normal distribution function, accounting for the fraction of microfacets oriented in a way which allows them to reflect light at the observer, with Θ_r indicating the roughness of the target surface:

$$D = D(\Theta_n, \mathbf{h}, \Theta_r) = \frac{\Theta_r^2}{\pi((\Theta_n \cdot \mathbf{h})^2(\Theta_r^2 - 1) + 1)^2}. \quad (4.28)$$

G is the geometry function. Microfacets can shield other microfacets from light, or occlude other microfacets from the observers point of view. Occluded or shadowed microfacets cannot contribute to the reflection, and the geometry function models this effect by computing the fraction of microfacets which are neither occluded nor shadowed:

$$G = G(\Theta_n, \mathbf{v}, k) = \frac{\Theta_n \cdot \mathbf{v}}{(\Theta_n \cdot \mathbf{v})(1 - k) + k} \quad (4.29)$$

We use k for direct lighting, since we model only one single laser light source and are not interested in indirect lighting:

$$k = \frac{(\Theta_r + 1)^2}{8}. \quad (4.30)$$

Since LiDAR sensors have very small beam divergence Φ_ψ , the respective solid angle Ω_i is very small. Also, due to the very small measurement aperture Φ_{D_R} , Ω_s is also very small. We thus assume the variation of the BRDF $\mathbf{f}_r$ to be negligible over these angles:

$$\bar{\mathbf{f}}_r \approx \frac{1}{\Omega_i \Omega_s} \int_{\Omega_i} \int_{\Omega_s} \mathbf{f}_r d\omega_o d\omega_i \approx \frac{1}{3} \sum_{k=1}^3 \mathbf{f}_r^{(k)}. \quad (4.31)$$

This approximation significantly reduces the computational cost, enabling an efficient and practical implementation of our LiDAR model.

The Cook-Torrance model has been developed for fast and realistic image rendering purposes, as have the material parameters in the CARLA simulator. Since we do not have the budget or time resources to change all material parameters in the CARLA simulator, we decide to use the material parameters as-is for our LiDAR simulation. Bolkas et al. [BM18] show that the color of a material in the visible spectrum is a good indicator for the reflectivity in the LiDAR spectrum. While they use LiDAR sensors with a wavelength of 1535 nm and 1550 nm for their experiments, we aim to simulate a HDL64 sensor with

a wavelength of 905 nm. This is much closer to the visible spectrum than in the tests conducted by Bolkas et al. Thus, we assume that the color of the material in the visible spectrum is a sufficient approximation for 905 nm. Gschwandtner et al. [Gsc+11] also rely on this assumption for *BlenSor*, but ignore geometry and only use the color of the target to determine reflectivity.

4.3.4 Threshold-Intersection Model

This component is designed to find the point in time t_x where the returning power coming back from the target and being picked up by the PD crosses the threshold Φ_{P_x} to trigger a measurement. As not every reflected pulse is strong enough to be detectable (cf. Fig. 4.5), this component deals with the “visibility dilemma”: A too large threshold Φ_{P_x} will result in very limited visibility range but good robustness against ghosting, i.e. no measurements will be triggered by background noise. A too low threshold Φ_{P_x} will result in high range visibility but will make the sensor more sensitive to spurious measurements, which can be caused by electrical measurement noise or external light sources such as sunlight.

In ideal conditions (no noise), for a given instance of a *LiDAR* model Φ with its geometric properties, and target properties Θ , the returning power P_R only varies across time t - the other parameters do not change. Thus, we can solve for t_x by doing binary search across t .

$$\begin{aligned} P_R(\Phi, \Theta, t_x) &= \frac{\pi \Phi_{D_R}^2}{4R^2} \Phi_{P_E} \cos \Theta_\theta \tilde{A}(\Phi, t_x) \tilde{f}_r(\Theta) \\ &\stackrel{!}{=} \Phi_{P_x} \end{aligned} \quad (4.32)$$

If the maximum power $P_{R_{\max}}$ of a returning pulse

$$P_{R_{\max}} = \max_t P_R(\Phi, \Theta, t) \quad (4.33)$$

is below the threshold Φ_{P_x} , the target is not visible to the sensor. Otherwise, there exists an intersection time t_x :

$$t_x = \begin{cases} \min_t |P_R(\Phi, \Theta, t) - \Phi_{P_x}| & \text{if } \Phi_{P_x} \leq P_{R_{\max}}, -\infty \leq t \leq 0 \\ \infty & \text{else} \end{cases}. \quad (4.34)$$

This represents the time, at which the leading edge crosses the threshold Φ_{P_x} . The falling edge t_f (i.e. when the signal drops below the threshold again) is detected in the same way.

Walk error Using the intersection of the threshold with the signal (i.e. t_x) to compute the range directly would lead to large walk errors, especially for slanted surfaces, since the slanted or darker surfaces basically “delay” the rise of the returning pulse power P_R at the receiver. In other words, the stronger the signal, the earlier the threshold Φ_{P_x} is crossed and the closer the estimated range R to the target would be. This effect is demonstrated

in Fig. 4.5, where we compare the echoes for targets with equal range and orientation, but different material properties. Without compensation, the timing error just caused by varying colors or orientation would be in the range of nanoseconds (1 ns timing error results in 0.15 m range error), which is unacceptable. It is thus necessary to perform walk error compensation. There are many methods to correct for walk error [CC17; Ins18; Yan+19; Sed+24]. We select the simple method by Amann et al. [Ama+01], since the actual method used by the manufacturer is not disclosed for any commercially available (automotive) LiDAR sensor. By comparing the difference in expected TDC clock ticks n_{expected} (i.e. the amount of TDC clock ticks that corresponds to the pulse width $\Phi_{t_{\text{pw}}}$) and the observed clock ticks n_{observed} between the leading and falling edge time t_x and t_f of the pulse, the “actual” time of flight $t_{\text{corrected}}$ can be estimated.

$$\begin{aligned} n_{\text{expected}} &= \left\lfloor \frac{\Phi_{t_{\text{pw}}}}{t_{\text{res,TDC}}} \right\rfloor \\ \Delta n &= n_{\text{observed}} - n_{\text{expected}} \\ t_{\text{corrected}} &= t_x + \frac{\Delta n}{2} t_{\text{res,TDC}} \end{aligned} \tag{4.35}$$

This ultimately leads to the measured range (assuming absence of noise):

$$\hat{R}_{\text{noise-free}} = \frac{c t_{\text{corrected}}}{2} \tag{4.36}$$

For more details on walk error LiDAR systems the interested reader is referred to [Li08; Kur12].

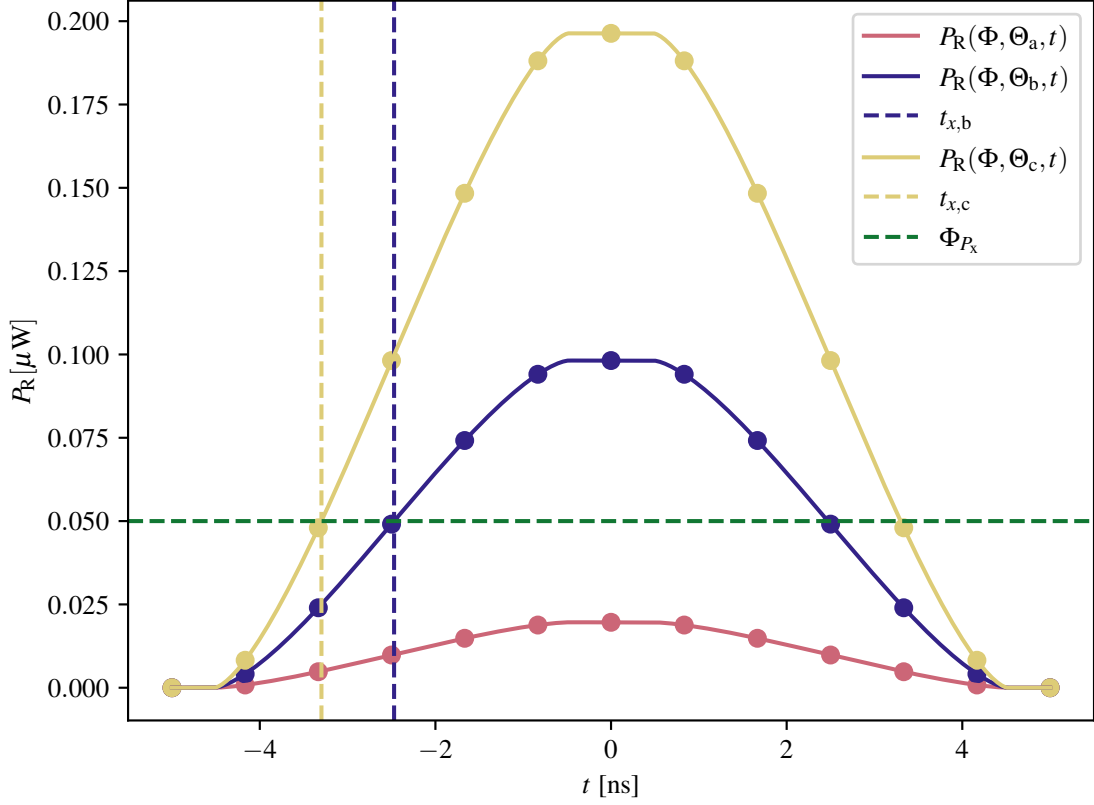


Figure 4.5: The returning signal P_R for two targets with identical geometric properties such as range and orientation, but with different material properties. The red returning pulse does not cross the threshold Φ_{P_x} and is too weak to be picked up. For the yellow and blue pulses, despite having equal range to the target, the threshold is crossed at different times $t_{x,a}$ and $t_{x,b}$. This is called **walk error**: If ignored, the weaker the signal, the later the threshold Φ_{P_x} is crossed and the “further” the estimated range to the target would be. By comparing the measured pulse width at the receiver with the pulse width of the laser emitter, the effect can be compensated. Note that the signal consists of discrete measurements, as demonstrated by the discrete markers on the signal curves. The low resolution is exaggerated for demonstration purposes.

4.3.5 Noise Model

Up to this point, the model does not take any noise into account. However, each measurement is subject to noise which needs to be modeled and is characteristic for the sensor. Step by step simulation for each signal through the whole signal chain allows us to add physically plausible noise to the simulation pipeline.

There are multiple sources of noise in a LiDAR system: Firstly, the high pulse frequency of the LiDAR causes variations in pulse power Φ_{P_E} , influencing measurement accuracy and the visibility of targets directly. For this, we assume Gaussian noise. Secondly, the range measurement is directly influenced by the time measurement error t_e , which

consists of the following parts: There is the uniformly distributed time quantization error $t_{e, \text{Discretization}}$ when converting the signal from continuous time to discrete time in the TDC. The clock mechanism that is part of the LiDAR is itself subject to Gaussian noise: $t_{e, \text{Clock}}$, see also [Alv+03; CC17]. Furthermore, as with all electronic measurement devices, there is also an amplitude jitter on the actual signal P_R , which influences the accuracy of the time measurement $t_{e, \text{Amplitude}}$.

Signal-to-Noise Ratio (SNR) is defined as the ratio of the mean power of a signal and the standard deviation of the noise (σ not to be confused with the backscattering cross-section σ discussed in Section 4.3.2):

$$\text{SNR} = \frac{\mu}{\sigma}, \quad (4.37)$$

which in our case leads to

$$\Phi_{\text{SNR}} = \frac{P_R}{\sigma_{\text{signal jitter}}}, \quad (4.38)$$

cf. [Sem23]. From literature [Li08; Wil18] we further know that

$$\sigma_{\text{timing jitter}} = \frac{\sigma_{\text{signal jitter}}}{\left. \frac{\partial P_R}{\partial t} \right|_{t=t_x}}. \quad (4.39)$$

This means, that the resulting timing jitter from signal jitter is inversely proportional to the gradient of the signal at the threshold crossing time t_x , i.e. the sharper the signal rises at the TDC, the less the signal jitter will influence the measurement. The slower the signal rises, the more impact the signal jitter will have on the measured time t_x at which the threshold Φ_{P_x} will be crossed, see also Fig. 4.6.

The measured range $\hat{R}$ is directly dependent on the measured time $\hat{t}$, which is subject to timing error t_e , which results in range error r_e :

$$\begin{aligned} \hat{t} &= t_{\text{corrected}} + t_e \\ \hat{R} &= \hat{R}_{\text{noise-free}} + r_e \\ r_e &= \frac{ct_e}{2} \end{aligned} \quad (4.40)$$

The following measurement errors influence t_e , which ultimately lead to the noisy, measured range $\hat{R}$ via $\hat{t}$:

$$\begin{aligned} t_e &= t_{e, \text{Discretization}} + t_{e, \text{Clock}} + t_{e, \text{Amplitude}} \\ t_{e, \text{Discretization}} &\sim U\left(-\frac{t_{\text{res, TDC}}}{2}, \frac{t_{\text{res, TDC}}}{2}\right)[\text{s}] \\ t_{e, \text{Clock}} &\sim \mathcal{N}(0, \sigma_{\text{clock}})[\text{s}] \\ t_{e, \text{Amplitude}} &\sim \mathcal{N}\left(0, \sigma_{\text{timing jitter}}\right)[\text{s}] = \mathcal{N}\left(0, \frac{\sigma_{\text{signal jitter}}}{\left. \frac{\partial P_R}{\partial t} \right|_{t=t_x}}\right)[\text{s}] \end{aligned} \quad (4.41)$$

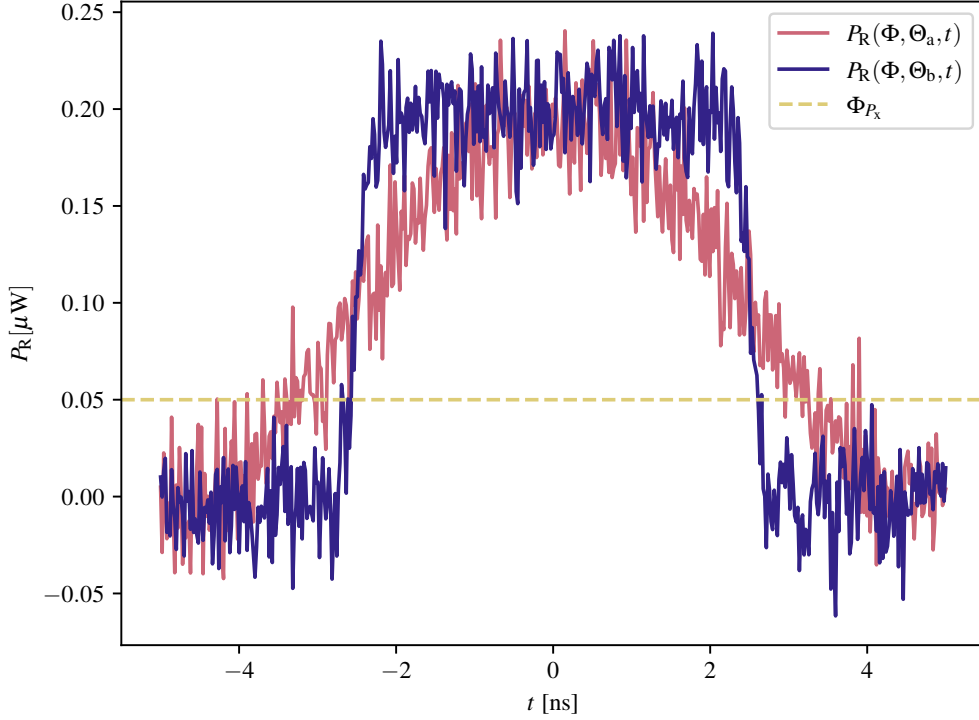


Figure 4.6: The returning echo signal P_R for two identical targets (except for angle of incidence Θ_θ) with added signal jitter and $\Phi_{SNR} = 10$. The signal jitter (jitter along vertical signal axis) is translated to timing jitter (i.e. error on the horizontal time axis). The steeper the signal rise, the less the timing error influences the measurement. Hence, in the example depicted here, the time t_x for target (b) with the steeper signal rise is measured more accurately than for target (a). Note that due to walk error compensation (cf. Fig. 4.5), the absolute elongation of the signals is not influencing the measurement accuracy.

4.3.6 Influence of Laser Layout on Raydrops

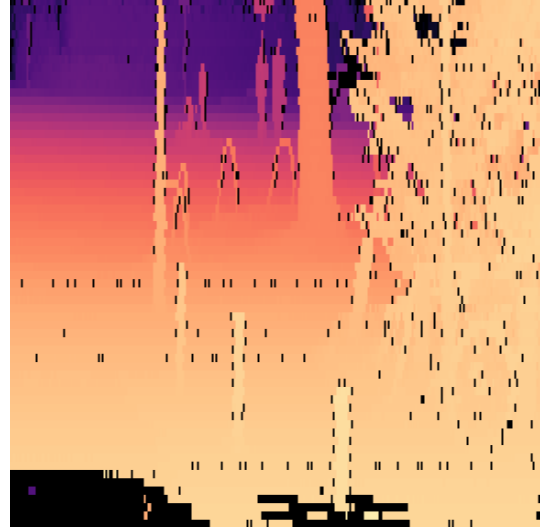
Upon inspection of real world LiDAR data projected to range images (cf. Fig. 4.7 and Fig. 4.11), we notice that the probability of a raydrop is not uniformly distributed over the layers of the sensor.

There is a characteristic saw tooth pattern in raydrops along the edges of objects, like poles or traffic lights, but also along the edges of cars, basically everywhere where the range changes abruptly between neighboring points along the layers. In an alternating pattern between layers, the probability of a raydrop is much higher when the left neighboring point is closer versus when the right neighboring point is closer or vice versa, leading to a sawtooth pattern along the edges of objects. This phenomenon occurs due to the layout of the laser emitters and receivers in the sensor and the resulting shadow effects, see Fig. 4.8.

In order to model this effect (occlusion due to sensor layout) in a simulator, we would need to model 64 individual light sources and their respective receivers at separate



(a) KITTI Camera Image with traffic light and small poles.



(b) KITTI range image with alternating ray-drops along the poles' height.

Figure 4.7: Example of raydrops on poles in the KITTI dataset.

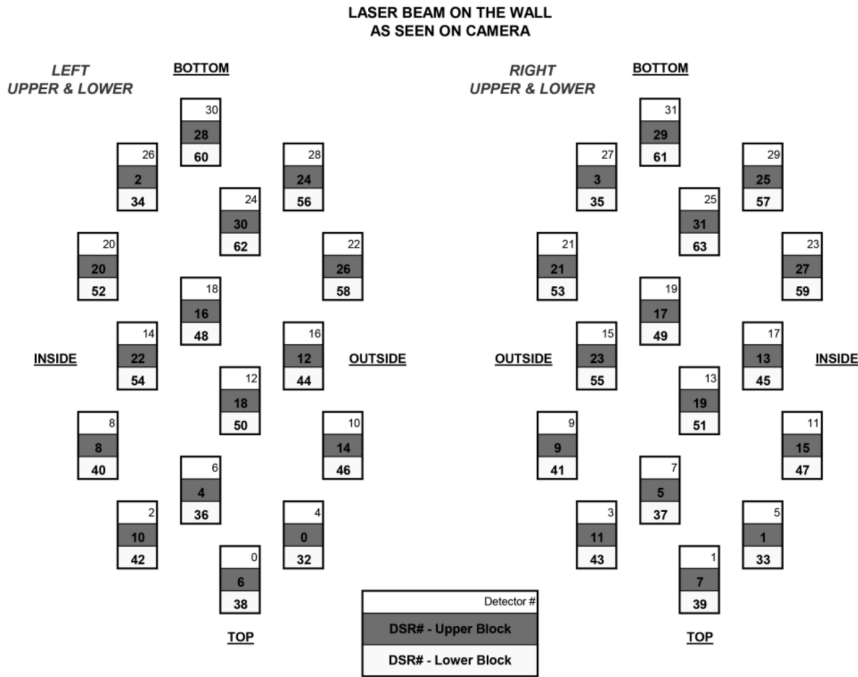


Figure 4.8: Orientations of the *laser* emitter and receivers in the sensor. Image extracted from sensor operating manual [Vel13]. The sensor consists of four *laser* blocks, upper left, upper right, lower left, lower right with 16 lasers each. Laser 0 and 32, 1 and 33, ... fire simultaneously while the sensor rotates.

geometric locations during the simulation, which would be prohibitively expensive. We

thus decide to take a probabilistic approach when modeling this effect, i.e. we compute the empirical probability of left and right raydrops for each layer, see Fig. 4.9.

To this end, we project the point clouds from real sensor data (from KITTI) into range images as described in [Tri+20], simultaneously keeping track of pixels that have no point projected into them, i.e. the raydrops. This gives us a range image $\mathbf{R} \in \mathbb{R}^{L \times C}$, where L is the number of layers and C the number of columns in the range image. Since not every pixel in the range image has a valid return, i.e. is non-empty, a boolean mask $\mathbf{V} \in \{0, 1\}^{L \times C}$, i.e. $V = \mathbf{V}^{(i,j)} = 0$ indicates whether there is a raydrop at layer $i \in L$ and column $j \in C$. Pixels in the range image $\mathbf{R}^{(i,j)}$ without a valid return, but with two valid left- and right neighbors are filled with the mean range of the neighbors ($\mathbf{R}^{(i,j-1)}$ and $\mathbf{R}^{(i,j+1)}$), but the boolean mask is kept untouched.

We then compute the range image gradients $\frac{\partial \mathbf{R}}{\partial j}$ along each layer (corresponding to range image row i) over all range images in the KITTI dataset. Analogous to image gradients, the gradient for a point $\mathbf{p}$ that falls into the range image at layer i and column j is computed as:

$$\begin{aligned} R(\mathbf{p}) &= \mathbf{R}^{(i,j)} = \|\mathbf{p}\|_2 \\ \frac{\partial R(\mathbf{p})}{\partial j} &= \frac{\partial \mathbf{R}^{(i,j)}}{\partial j} = \frac{\mathbf{R}^{(i,j+1)} - \mathbf{R}^{(i,j-1)}}{2}. \end{aligned} \quad (4.42)$$

The column index has no influence on the probability of a raydrop, since a range image is a 2D projection of a 3D point cloud, and as the sensor rotates, the resulting column index is arbitrary. For each layer i , we compute the total number of pixels that have a certain range R , range gradient $\frac{\partial R}{\partial j}$ and two valid neighboring pixels, and the number of dropped pixels that have a certain range R , range gradient $\frac{\partial R}{\partial j}$ and two valid neighboring pixels.

Averaging over all range image columns C of all the N point clouds of the dataset, giving us the empirical probability of a raydrop $p_d(R, \frac{\partial R}{\partial j}, i)$ for a point $\mathbf{p}$ falling into a given layer i with range image gradient $\dot{R}_l < \frac{\partial R}{\partial j} \leq \dot{R}_u$ and range $R_l < R \leq R_u$.

$$\begin{aligned} P(V = 0 \mid R_l < R \leq R_u, \dot{R}_l < \frac{\partial R}{\partial j} \leq \dot{R}_u, i) &= \\ \frac{\sum_{n=0}^{N-1} \sum_{j=1}^{C-2} \mathbb{I}[R_l < R \leq R_u] \mathbb{I}[\dot{R}_l < \frac{\partial R}{\partial j} \leq \dot{R}_u] \mathbb{I}[\mathbf{V}^{(i,j-1)}] \mathbb{I}[\neg \mathbf{V}^{(i,j)}] \mathbb{I}[\mathbf{V}^{(i,j+1)}]}{\sum_{n=0}^{N-1} \sum_{j=1}^{C-2} \mathbb{I}[R_l < R \leq R_u] \mathbb{I}[\dot{R}_l < \frac{\partial R}{\partial j} \leq \dot{R}_u] \mathbb{I}[\mathbf{V}^{(i,j-1)}] \mathbb{I}[\mathbf{V}^{(i,j+1)}]} \end{aligned} \quad (4.43)$$

Example statistics for (4.43) over the whole KITTI dataset (from 5 m to 10 m for an absolute range gradient threshold of 0.5 m and up) can be seen in Fig. 4.9. We notice, that the probability of a raydrop is not uniformly distributed over the layers of the sensor, but that there is a distinct alternating pattern due to the pattern of laser emitters and receivers, see Fig. 4.8.

In the raw and unprocessed data from the simulator, there are no missing returns. Hence, at inference time, we can then compute the range gradient $\frac{\partial R}{\partial j}$ for each pixel in

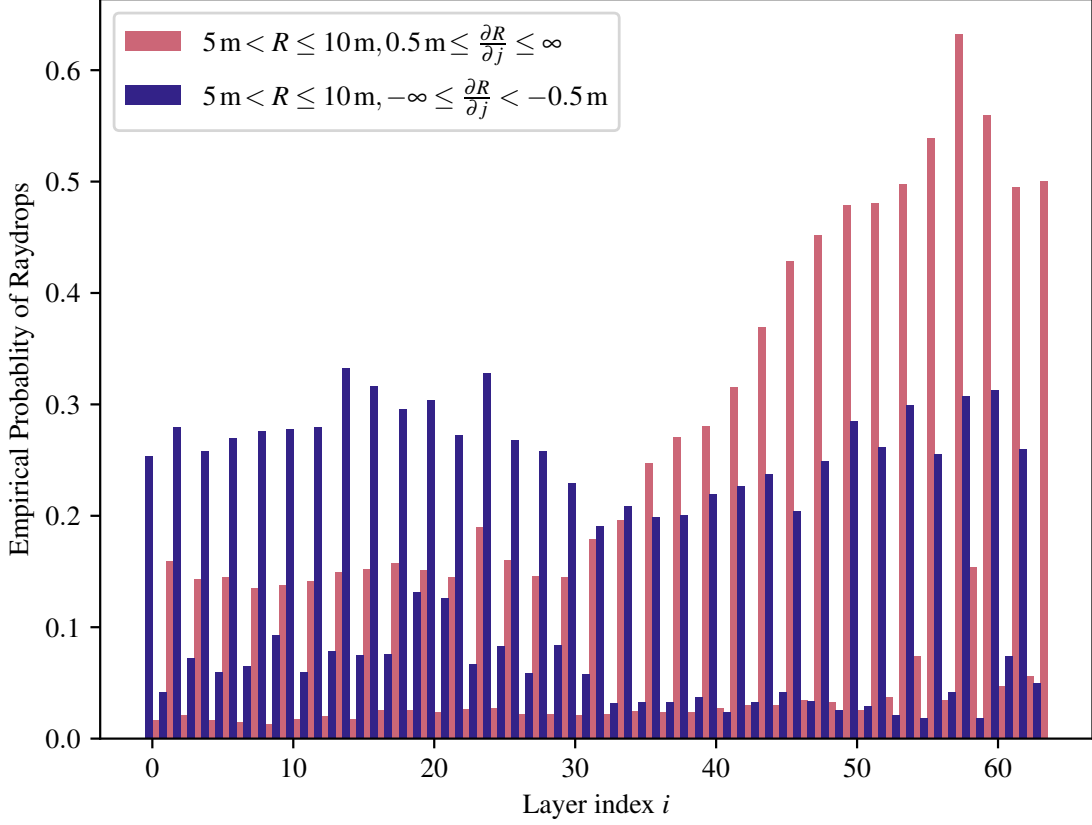


Figure 4.9: Empirical probability for raydrops on KITTI depending on "left" ($\frac{\partial R}{\partial j} < -0.5$ m) and "right" ($\frac{\partial R}{\partial j} \geq 0.5$ m) raydrops for each layer from range R 5 m to 10 m. Note: Layer index 0 refers to the "top" layer, i.e. the one with the highest elevation.

the simulated range images, draw a random number, and use the empirical probability $P(V = 0 \mid R_l < R \leq R_u, \dot{R}_l < \frac{\partial R}{\partial j} \leq \dot{R}_u, i)$ to determine whether a point is to be dropped or not, the result of which can be seen in Fig. 4.10, where we also compare the results of our physical LiDAR simulator with [BlenSor](#). The effect of alternating raydrops makes our model appear more realistic than the [BlenSor](#) model, especially when compared to the real world data in Fig. 4.11. At inference, the empirical probability is stored in a look-up table, with range R ranging from 0 m to 120 m and range gradient $\frac{\partial R}{\partial j}$ ranging from -120 m to 120 m. The bin size of the lookup-table is set to 5 m for range R , and 1 m for the range gradient $\frac{\partial R}{\partial j}$.

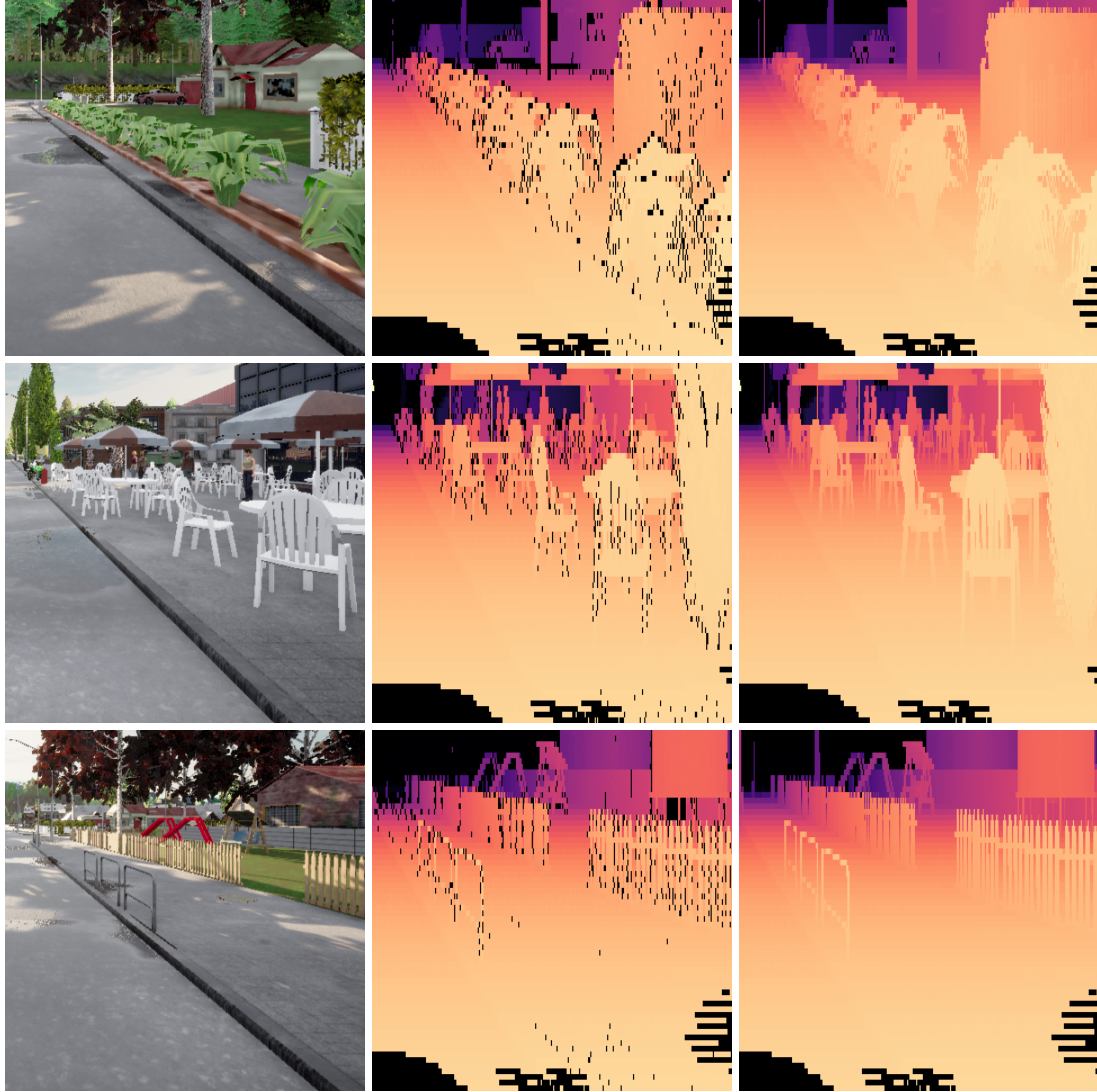


Figure 4.10: Example of range images with raydrops for our physical *LiDAR* simulator (center) and *BlenSor* (right). Left column: Camera image for context. Range images are colored using the same methodology from Fig. 2.3b, with the vertical resolution being increased fivefold for better visibility. Our physical *LiDAR* simulator produces raydrops with a similar pattern as in the real world data in Fig. 4.11, while *BlenSor* does not.

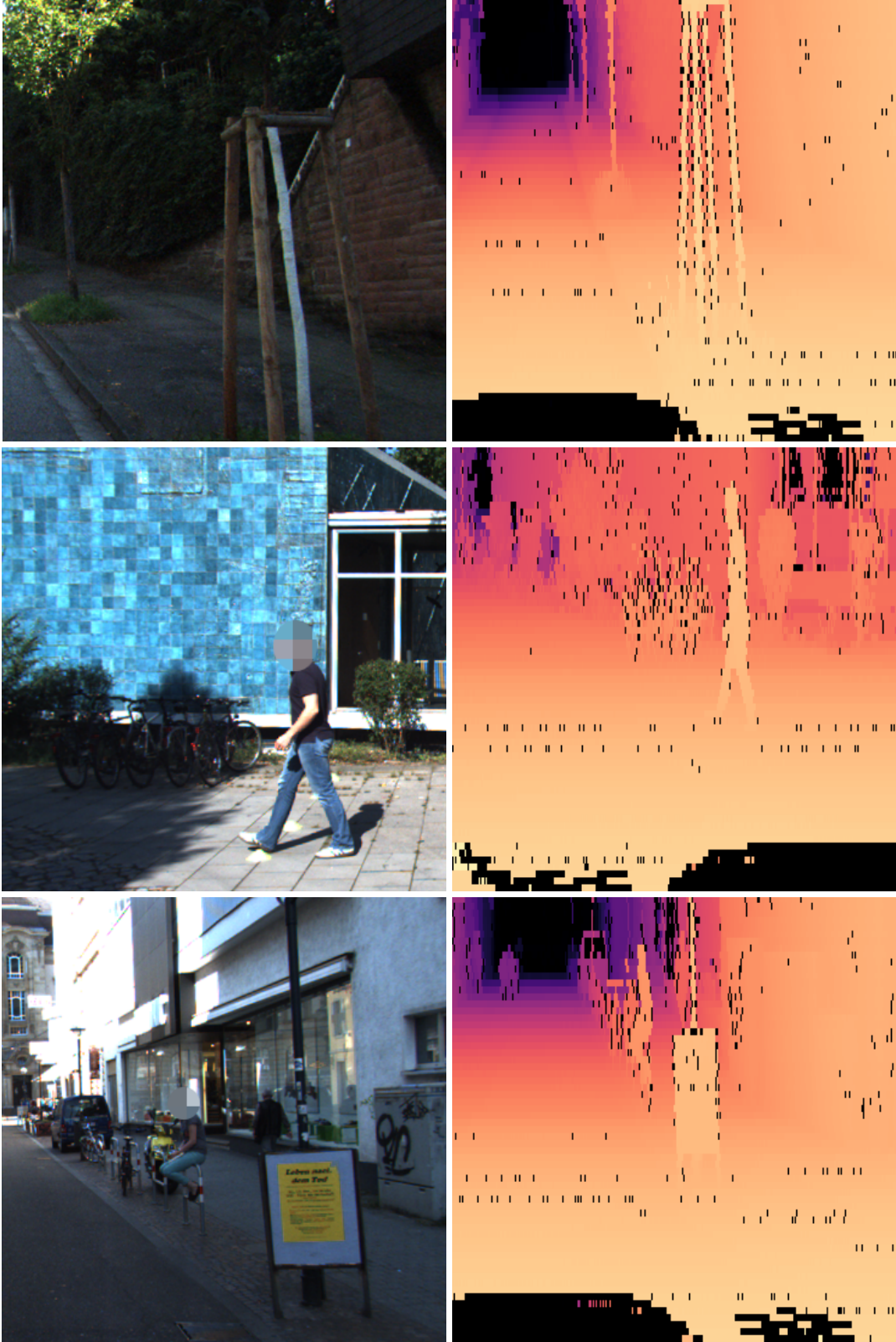


Figure 4.11: Example of raydrops in the KITTI dataset with alternating pattern along the edges of objects and the camera image for context. The range images are computed from the point clouds using the method described in [Tri+20], and vertical resolution is increased fivefold for better visibility.

4.4 Parameter Selection

In Table 4.2 we list and briefly explain the parameters we have set for our physical model, as well as the reasoning for our specific parameter choices. In general, we tried to keep the parameters as close as possible to what has been reported in the literature for the HDL64 sensor [Vel13]. For parameters where no information was available, we used values from relevant literature. We only deviated from the reported values when the reported parameters produced unrealistic results in our simulator. This was the case for the laser power, which we set to 100 W instead of the reported 70 W, since the latter produced too many raydrops in our simulator. One possible explanation for this discrepancy could be that the material properties of the objects in the CARLA simulator might differ from the real world, leading to a higher absorption of the laser light.

Parameter	Symbol	Value	Source
Beam divergence	Φ_ψ	2 mrad	extracted from [GL10]
Pulse width	$\Phi_{t_{pw}}$	5 ns	extracted from [GL10]
Emitter power	Φ_{P_E}	100 W	extracted from [Sem23]
Signal-to-noise ratio	Φ_{SNR}	10	extracted from [Sem23]
Std. dev. of emitter power	$\Phi_{\sigma_{P_E}}$	3 W	based on visual tuning
Visibility power threshold	Φ_{P_x}	10 nW	based on visual tuning
Base reflectivity (dielectrics)	Θ_{f_0}	0.04	default of [Vri24]
TDC resolution	$t_{res,TDC}$	55 ps	extracted from [CC17]
Clock noise	$t_{e,Clock}$	55 ps	[Alv+03], slightly visually adjusted
Laser emitter diameter	Φ_{D_L}	0.02 m	estimated after inspecting the sensor emitter lens system
Receiver diameter	Φ_{D_R}	0.05 m	estimated after inspecting the sensor receiver lenses system

Table 4.2: The parameters we have selected for our physically-based simulation model.

4.5 Implementation Details

While CARLA [Dos+17a] does offer a LiDAR implementation based on ray casting, we use our own custom implementation based on a custom "camera", since at the time of this work, the ray casting based sensor was only giving very rough hitbox intersections, and not the actual mesh intersection, cf. Fig. 4.12.

Our implementation is based on a rotating depth camera, since this enables us to access additional material properties such as roughness, metallicness and surface normals, which the ray casting-based sensor cannot expose, due to these attributes being only accessible from the GPU shaders. We update the simulation (ego-vehicle position and position of all actors in the scene) 500 times per second, i.e. 50 times for one revolution of our virtual LiDAR sensor, which does one 360° revolution each 0.1 s. In terms of which pixel

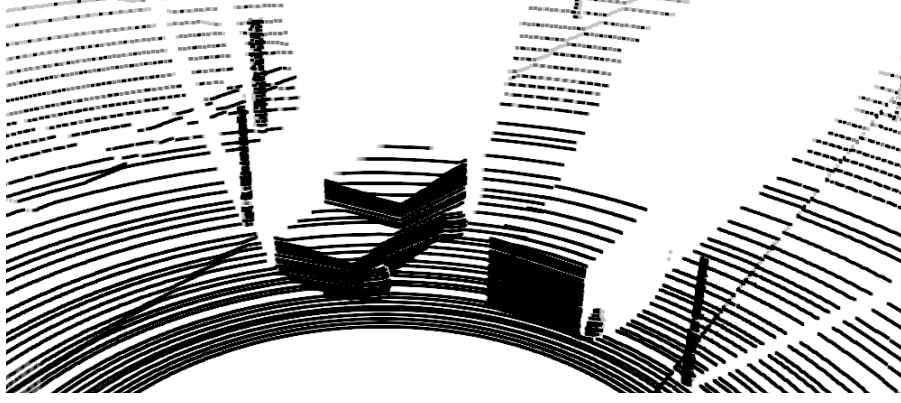


Figure 4.12: Point cloud with box artifacts from the default LiDAR in CARLA at the time of writing this chapter. The fine geometry including the surface normal, material parameters such as roughness are only accessible from GPU, hence we use a z-buffer (GPU) based implementation. Image extracted from <http://asrl.utoronto.ca/datasets/mdlidar/index.html>, accessed on 22nd of June 2024. Modifications applied: Colors inverted, morphological erosion for better quality in print.

locations' data to extract from the z-buffer, we follow [HCW19]: Knowing the vertical FOV and horizontal resolution of the LiDAR sensor from the manufacturers datasheet [Vel13], we project the individual LiDAR rays into the respective locations in the z-buffer. From this location in the z-buffer, we gather all the data we need for our simulation:

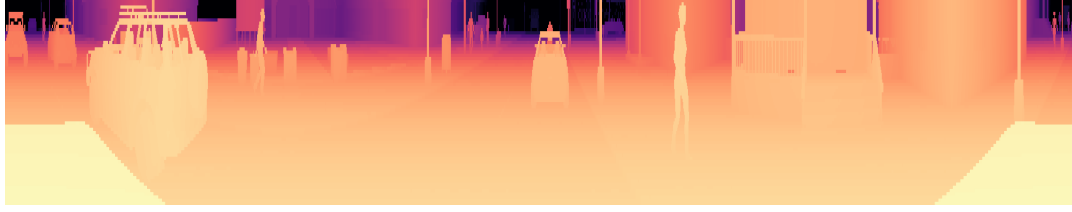
1. Range R
2. Surface Normal Θ_n
3. Surface Roughness Θ_r
4. Surface Albedo (Matte Surface Color) Θ_{rgb}
5. Surface Metallicness Θ_m
6. Semantic Segmentation (only for visualization)

An example point cloud and corresponding attributes projected in to a range image can be seen in Fig. 4.13.

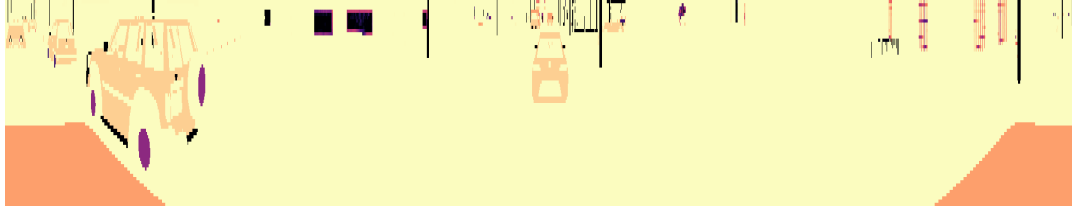
Semantic Colormap In Fig. 4.13 and Fig. 4.14 we show the semantic segmentation images from the KITTI dataset and the CARLA simulator with the following colormap:

- | | | |
|----------------|-------------------|----------------|
| • ■ Unlabeled | • ■ Other-vehicle | • ■ Building |
| • ■ Car | • ■ Person | • ■ Vegetation |
| • ■ Bicycle | • ■ Road | • ■ Terrain |
| • ■ Bus | • ■ Sidewalk | |
| • ■ Motorcycle | • ■ Other-ground | |

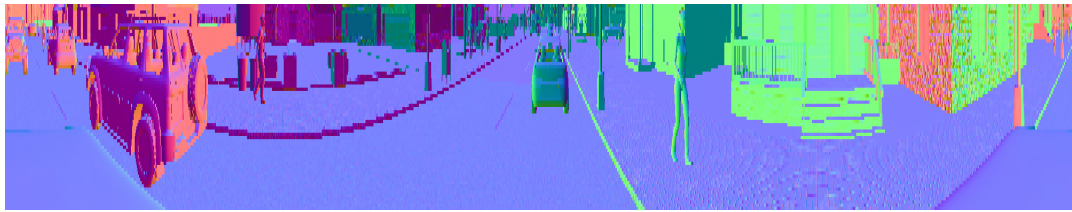
4 Physically-based *LiDAR* Simulation



(a) Range (Clipped to 50 m, normalized, color mapping according to Fig. 2.3b)



(b) Metallicness (Normalized, color mapping according to Fig. 2.3b)



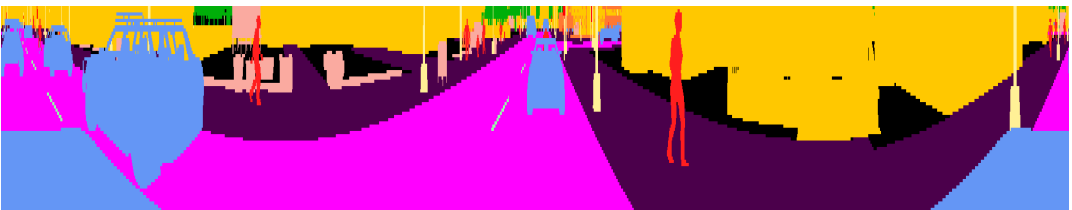
(c) Surface Normals, shown as RGB.



(d) (Matte) Base Color (RGB)

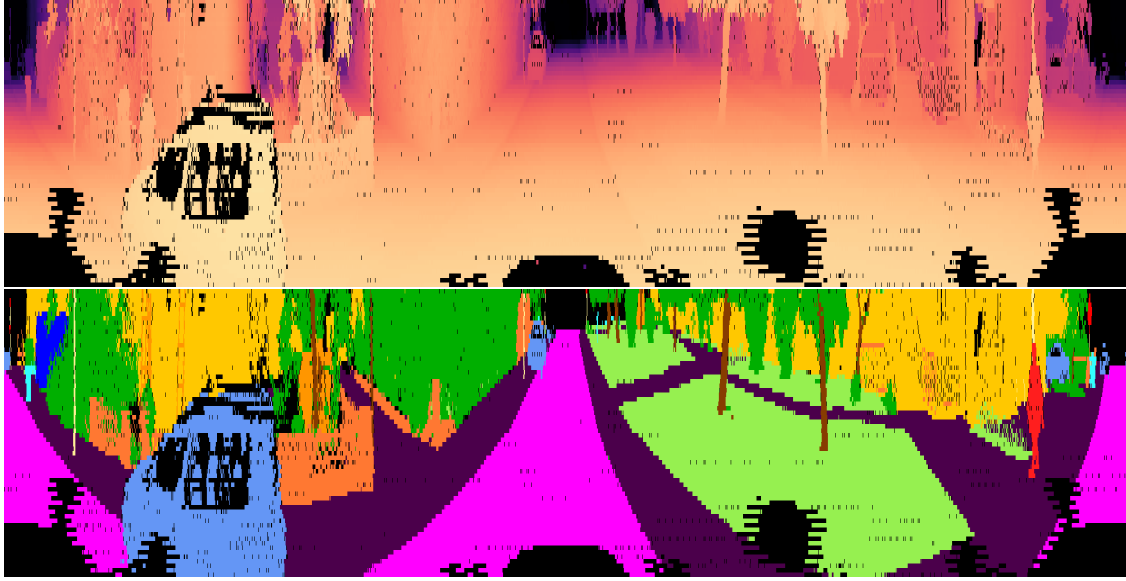


(e) Roughness (Normalized, color mapping according to Fig. 2.3b)

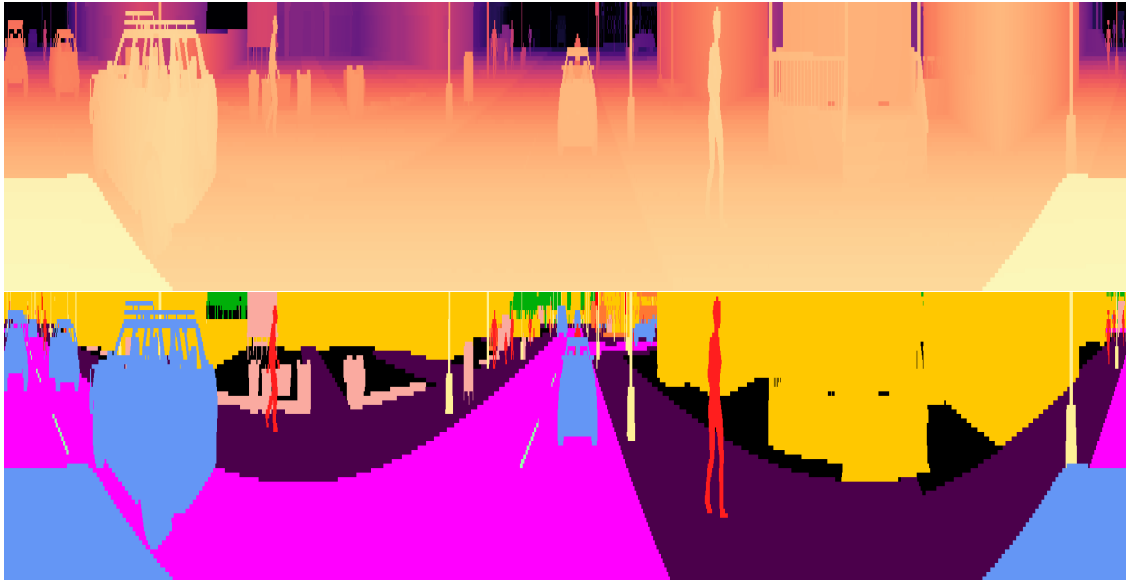


(f) Semantics, color mapped according to Section 4.5

Figure 4.13: All raw data channels exposed by CARLA to our simulator.



(a) KITT data



(b) Plain CARLA data

Figure 4.14: Range image and semantic segmentation image from SemanticKITTI and the Plain simulator data, without any noise or visibility simulation applied. The semantic colormap is given in Section 4.5. Range image colormap according to Fig. 2.3b.

4.6 Evaluation

We want to compare the performance of our physically-based simulation model to the baseline methods in Section 4.6.1 in terms of realism. We start with a visual comparison of the simulated data to real-world data, see Section 4.6.2. Furthermore, we evaluate the simulation models in three other aspects that we believe to be indicative of realism.

Firstly, we compare statistical properties related to noise and visibility range of the simulated data to real-world data, see Section 4.6.3.

The second evaluation method is based on the assumption that the more realistic a LiDAR simulation model is, the smaller the resulting domain gap should be for downstream tasks trained on this data when being applied to real data. In other words, the more realistic the simulated data, the easier it should be to transfer learned features to real data. So, we use simulated data as training data for a downstream task, and measure the performance on real data, see Section 4.6.4.

The third evaluation evaluates the usage of simulated data as validation tool, see Section 4.6.5. If a LiDAR model produces realistic measurements, perception models such as object detectors trained on real data should perform identically on previously unseen, but simulated data.

4.6.1 Baselines

We integrate all of our baselines into the CARLA simulator [Dos+17a], so that all the comparisons in this section are based on exactly the same source data. We compare our simulator to the following baselines:

Plain

This is essentially the clean, i.e. unaltered range data extracted from the simulator cf. Fig. 4.13, without any noise or visibility threshold applied. This represents a perfect LiDAR sensor that has unlimited range and no noise.

BlenSor

We use the original BlenSor [Gsc+11] implementation as published by Gschwandtner et al. [Gsc12] and apply it to the data generated by the CARLA simulator. As described in Section 3.1, BlenSor assumes noise to be Gaussian distributed and applies a visibility threshold based on the color and range of the target, cf. Fig. 3.1.

We chose 0.01 m as standard deviation for the range measurement, i.e. the provided default by BlenSor [Gsc12]. This is half the amount claimed by the manufacturer in the datasheet [Vel13], but it visually matches the actual noise distribution much better, as shown in Fig. 4.15.

4.6.2 Visual Comparison with Real-World Data

In the following, we give a visual comparison of point clouds generated by the different simulation models to real-world data from the KITTI dataset [GLU12b]. We show paired scenes, where the same scene is shown from the same perspective in the different simulators and for real data, see Fig. 4.16, Fig. 4.17, Fig. 4.18, Fig. 4.19 and Fig. 4.20.

When looking at the cars in Fig. 4.16 and Fig. 4.17, we notice that the Plain model produces very clean looking point clouds, while BlenSor and our physical model produce

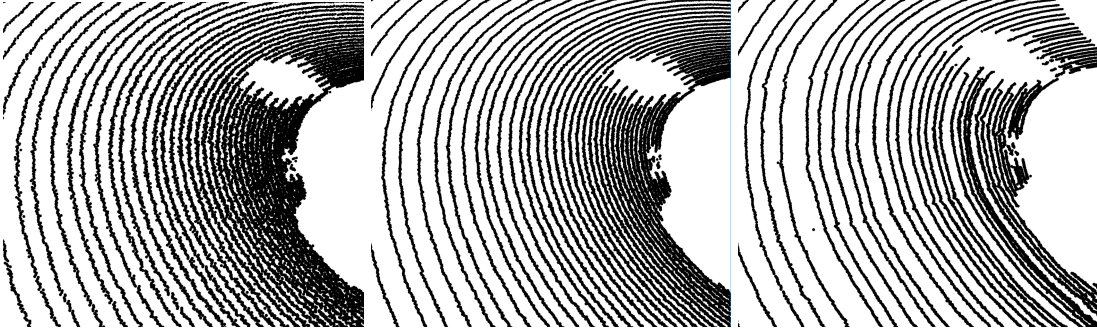


Figure 4.15: Comparing the range noise for *BlenSor* with standard deviation of 0.02 m (left) and 0.01 m (middle) with real KITTI data (right). According to the sensor datasheet [Vel13], the standard deviation of the range should be 0.02 m, but the scanlines close to the vehicle are almost blended together for the *BlenSor* simulation with 0.02 m, so we decide to use 0.01 m, which is also the default provided by [Gsc12] for HDL64 for a fair comparison.

more realistic looking point clouds. On cars, the difference in visibility models and noise is most pronounced. Our model produces the most realistic looking point clouds, with the noise pattern being most similar to the real data, as *BlenSor* and the Plain model produce much too clean looking point clouds.

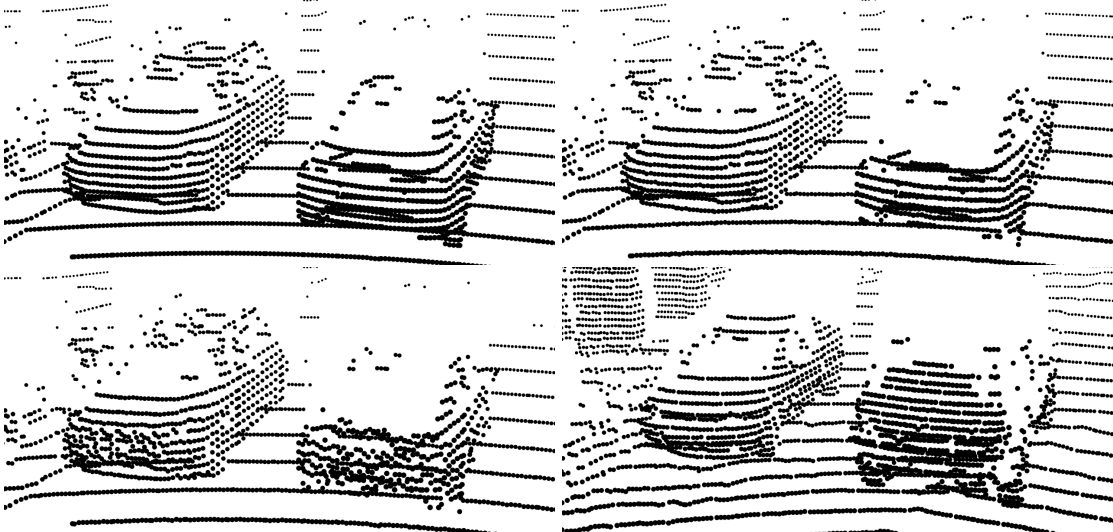


Figure 4.16: Comparing point clouds of cars with no range noise (Plain, upper left) with *BlenSor* (top right), with our physical model (bottom left) and real KITTI data (bottom right). The plain point clouds look most unrealistic compared to the real data, while *BlenSor* and our physical model look more realistic. *BlenSor* has a more uniform noise pattern, and the overall shape of the cars is still relatively well-preserved. Our physical model exhibits the strongest noise. The cars are still recognizable, but the noise is more pronounced.

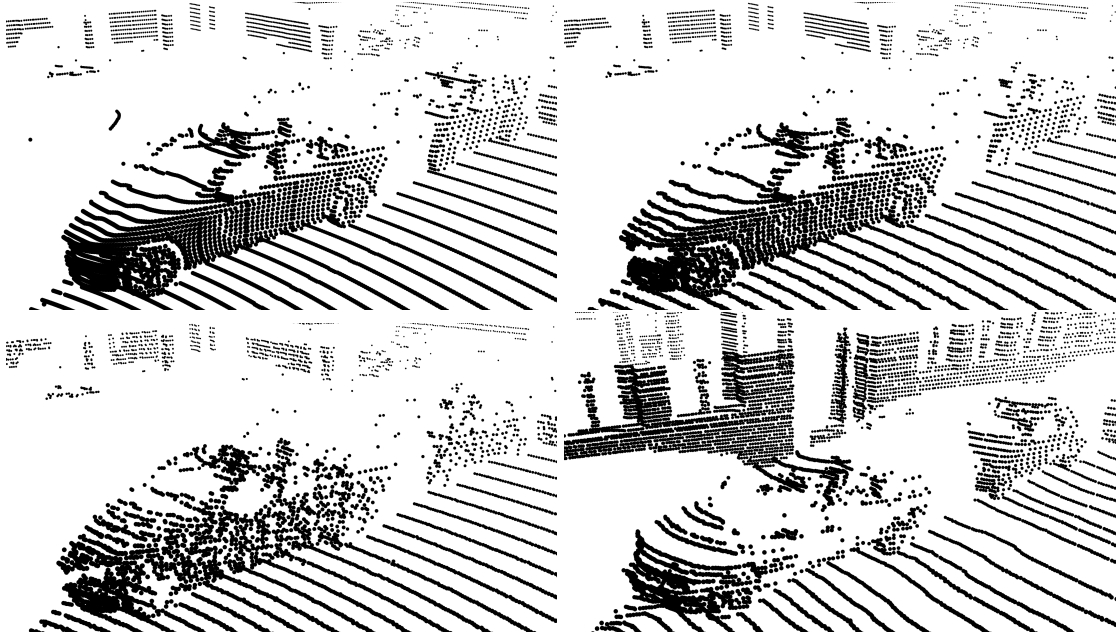


Figure 4.17: Comparing point clouds of cars with no range noise (Plain, upper left) with [BlenSor](#) (top right), with our physical model (bottom left) and real KITTI data (bottom right). Note how on the car door in the real KITTI data, returns are missing, as indicated by the white spots. Our physical model comes closest to the real data, as both [BlenSor](#) and the Plain model produce much too clean looking point clouds. [BlenSor](#) and the plain data do not exhibit any missing returns, while our physical model does. This, in combination with the noise pattern, makes our physical model look more realistic.

For points that fall on the road (see Fig. 4.18), the Plain simulated data, when compared to real-world data, shows that the ground surface in the simulation is perfectly flat. This effect is less noticeable in the point clouds produced by [BlenSor](#) and our physical model, which resemble the real KITTI data more closely because these models introduce some noise to the range measurements. However, based solely on points falling on the road, it is challenging to determine whether [BlenSor](#) or our physical model generates more realistic-looking point clouds.

On pedestrians, all simulation models produce point clouds that look quite different from the real data, see Fig. 4.19 and Fig. 4.20, with the Plain model producing slightly less noisy point clouds on pedestrians compared to the other models.

In this section, we visually compared point clouds generated by different simulation models to real-world data from the KITTI dataset. The comparison revealed that the Plain model produces overly clean point clouds, while [BlenSor](#) and our physical model generate more realistic results by introducing noise. For cars, our physical model is the most realistic, mimicking the noise patterns seen in real data. The perfect flatness of the road surface in the Plain model’s point clouds contrasts heavily with the more realistic noise in [BlenSor](#) and our physical model. However, for pedestrians, all models appear smoother than real data, with the Plain model being the least realistic. Overall, our physical model

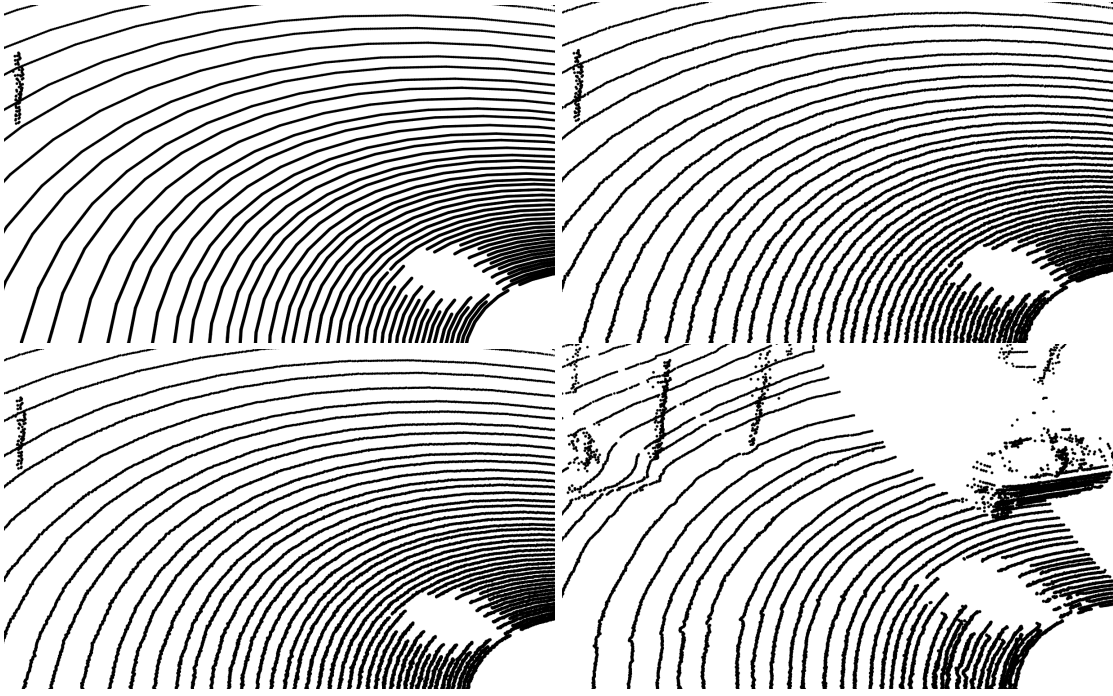


Figure 4.18: Comparing no range noise (Plain, upper left) with *BlenSor* (top right), with our physical model (bottom left) and real KITTI data (bottom right) of a road surface. Note that for the simulated point clouds we have artificially added the shadow of the recording setup from KITTI: Points in the range image that have never had a valid return over the whole KITTI dataset are masked out from the point cloud. While the Plain simulator has no added noise, some small simulation artifacts are visible in the point cloud, due to the way we extract the data from the z-buffer of the simulator. Like for the Plain simulated data, *BlenSor* and the physical model have near-perfect circular patterns in the point cloud, but visually look much more like the real KITTI data.

generally produces the most realistic point clouds, especially for cars.

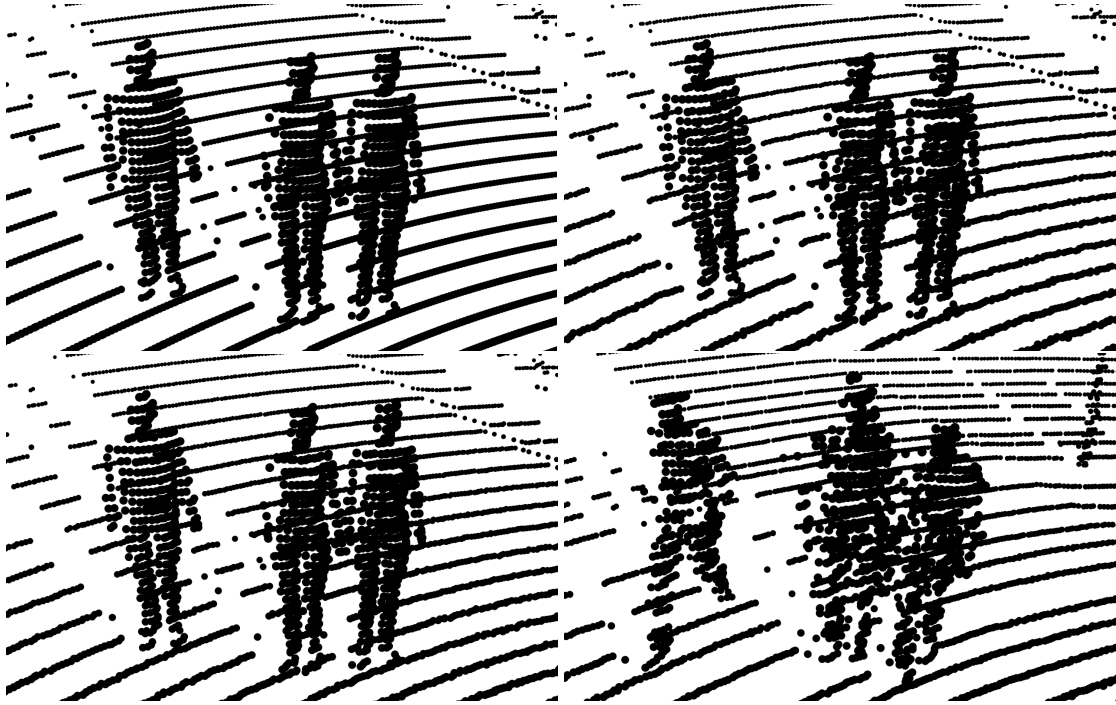


Figure 4.19: Comparing point clouds of pedestrians with no range noise (Plain, upper left) with *BlenSor* (top right), with our physical model (bottom left) and real KITTI data (bottom right). The difference between all simulator models is less pronounced for pedestrians than for cars. The plain point clouds look most unrealistic compared to the real data, while *BlenSor* and our physical model look more realistic, as they exhibit more noise. However, compared to the real data, the pedestrians look too smooth for all simulation models.

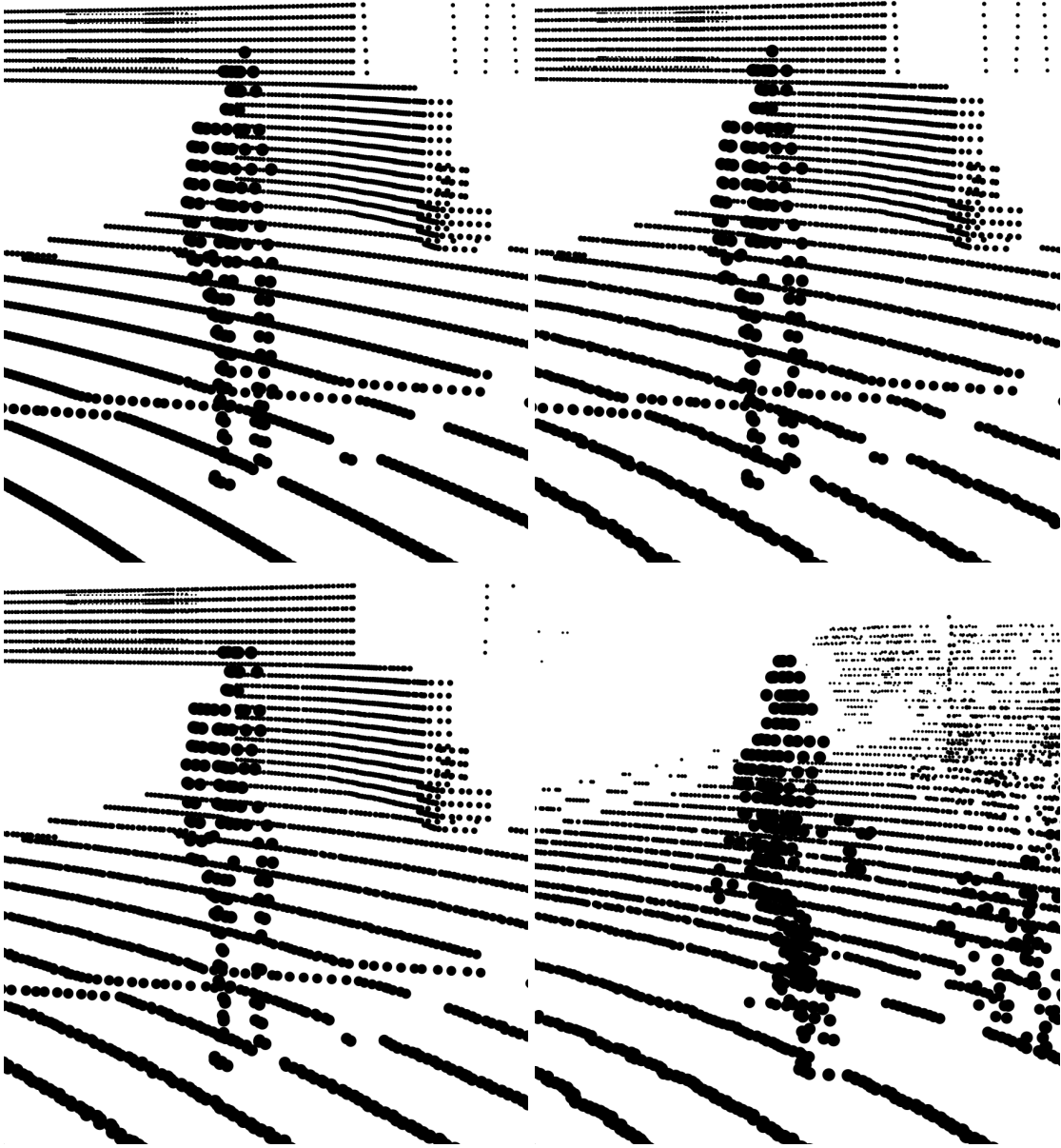


Figure 4.20: Comparing point clouds of a single pedestrian with no range noise (Plain, upper left) with *BlenSor* (top right), with our physical model (bottom left) and real KITTI data (bottom right). The difference between all simulator models is rather small, with the Plain simulator looking most unrealistic, and *BlenSor* and our physical model looking more realistic. However, the difference between *BlenSor* and our physical model is less pronounced for pedestrians than for cars.

4.6.3 Statistical Comparison with Real-World Data

To compare statistical properties of the different simulation models to a real-world sensor, we use the SemanticKITTI dataset [Beh+19b]. The SemanticKITTI dataset consists of a large part of the KITTI dataset, where the points from the HDL64 sensor have been annotated with their semantic classes.

Influence of Range on Point Density

Firstly, we compare influence of the measurement range on the amount of points that have been recorded, cf. Fig. 4.21. To this end, a histogram counting the number of points that fell into the different range bins (1 m intervals) is created for each data source. We have randomly selected 1000 point clouds from the real data and 1000 random point clouds from each of the simulation models. It is important to note that the simulated point clouds are all from identical scenes. While comparing absolute values between simulated and real data is challenging due to the less cluttered scenes in CARLA compared to KITTI, the comparison among simulation models and overall trends compared to real-world data remains valuable. This helps us analyze the different characteristics of the simulation models.

Looking at Fig. 4.21 we notice, that up to a range of around 40 m, the physical model yields a similar amount of points as the real sensor, while the *BlenSor* and Plain model produce more points. This is no surprise, since the Plain model does not apply any visibility threshold, and thus represents the idealized upper bound of the amount of points that can be recorded. The *BlenSor* model only starts to apply a visibility threshold at 40 m for very dark objects (see also Fig. 3.1), which is where the two curves (Plain, *BlenSor*) begin to diverge in Fig. 4.21. Our physical model seems to match the actual visibility characteristics quite well up to 50 m, and seems to slightly overestimate the amount of targets that are visible. At around 50 m, *BlenSor* matches the trend of the real sensor a bit better than our physical model, but drops too quickly below the real sensor, while the physical model stays between the Plain model and the real sensor. We also observe that the *LiDAR* data in the SemanticKITTI dataset seems to have been truncated at around 80 m, since the datasheet of the sensor reports a maximum range of 120 m [Vel13], but we only see data up to around 80 m in the dataset.

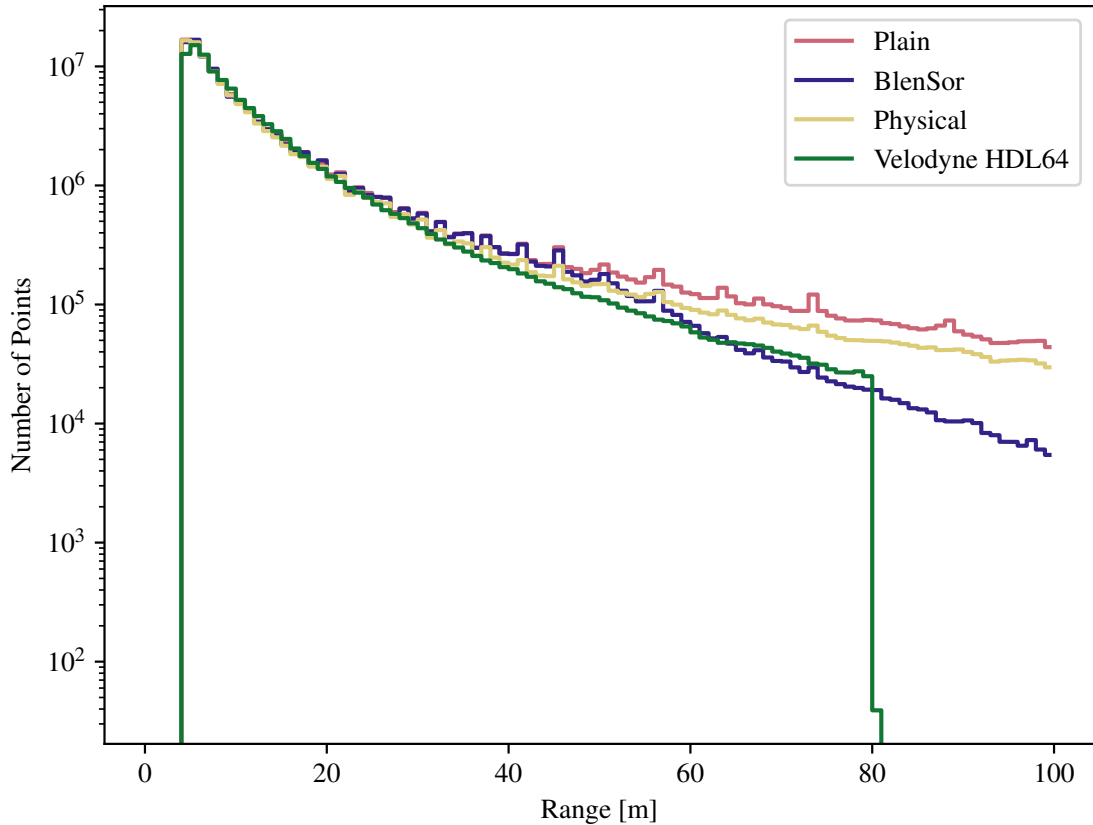


Figure 4.21: Number of points that have been recorded for 1000 point clouds in the KITTI dataset and the respective simulators. Note that the raw data for the simulated point clouds is identical, but the KITTI data is from unmatched scenes.

Influence of Range on Road-Point Density

As the point clouds from the SemanticKITTI dataset have been annotated with pointwise semantics up to a range of 50 m, the second comparison is between the amount of points falling on the road over increasing range, see Fig. 4.22. Comparisons to real data are thus only possible up to a range of 50 m.

In Fig. 4.22 we see that the simulated data exhibits distinct and periodic peaks in the histogram of point counts over range, which is especially pronounced for the Plain model. This occurs because of the perfectly flat road surface and the absence of vehicle dynamics in simulation. In the real world, the ego vehicle experiences roll and pitch due to acceleration, braking, and cornering, causing individual [laser](#) layers of the [LiDAR](#) sensor to fall into more distributed range bins. In the simulation however, the lack of these vehicle dynamics results in pronounced peaks in the distribution.

Similar to Fig. 4.21, we see that for [BlenSor](#) the visibility aspect only becomes relevant above 40 m, since it matches the statistics of the Plain simulator exactly up to this point. Even above that, up to around 55 m, the [BlenSor](#) model produces almost the same amount of points as the Plain model. [BlenSor](#) seems to overestimate the range of the sensor for

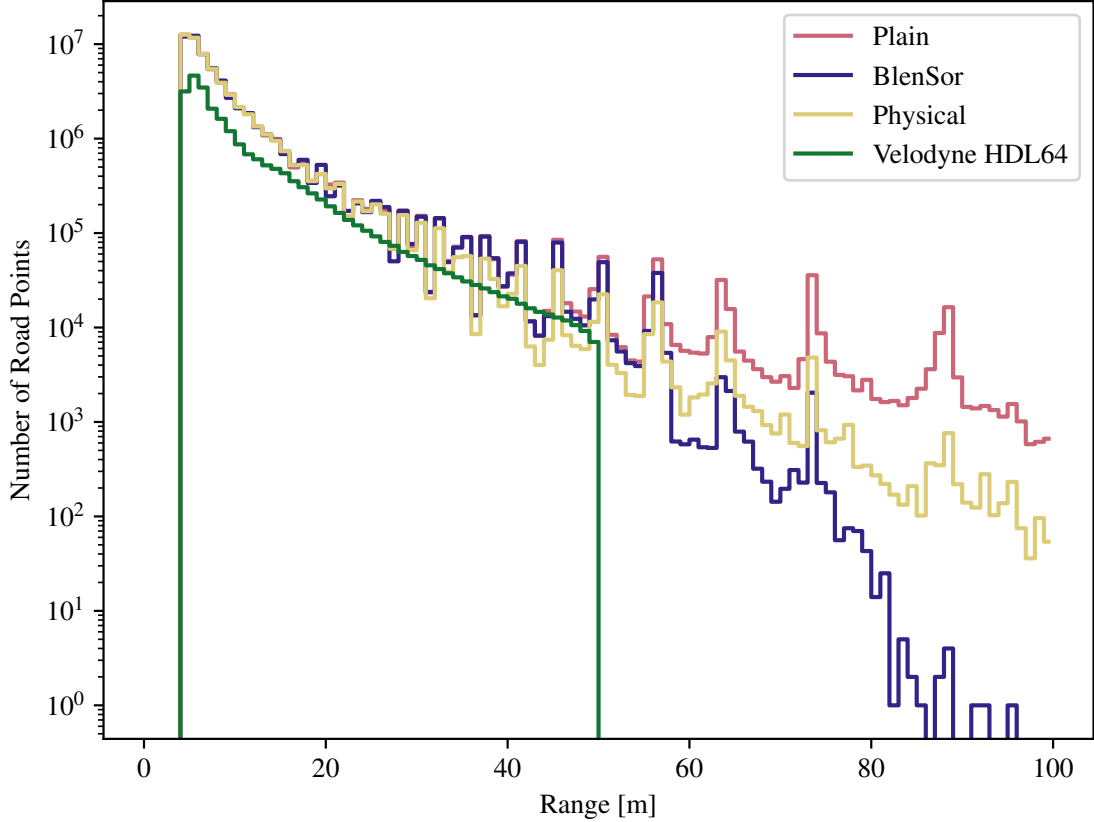


Figure 4.22: Number of points that fell on the road which have been recorded in 1000 point clouds in the KITTI dataset and the respective simulators (range bin size: 1 m). Note that the raw data for the simulated point clouds is identical, but the KITTI data is from unmatched scenes. The SemanticKITTI dataset features dense annotations up to 50 m. Beyond this range, valid comparisons cannot be made.

points on the road. On the other hand, we see that the curve representing the physical model drops earlier and stronger than the curve for the actual KITTI data, underestimating the amount of visible points on the road for 40 m and above.

Estimation of Measurement Errors on the Road Surface

Beyond the comparison of point densities at different range, we are also interested in comparing the characteristics of the noise on the range measurement.

Normally, this would require recording of paired data from the same scene both in simulation and in the real world or in a staged laboratory environment with calibrated targets. This is not feasible, since we do not have access to the original sensor or the resources to model the real world in the simulator. Instead, we use the SemanticKITTI dataset [Beh+19b] to compare the noise characteristics on a at least locally *known* flat surface: the road surface. For both simulation and real-world data we know the location of

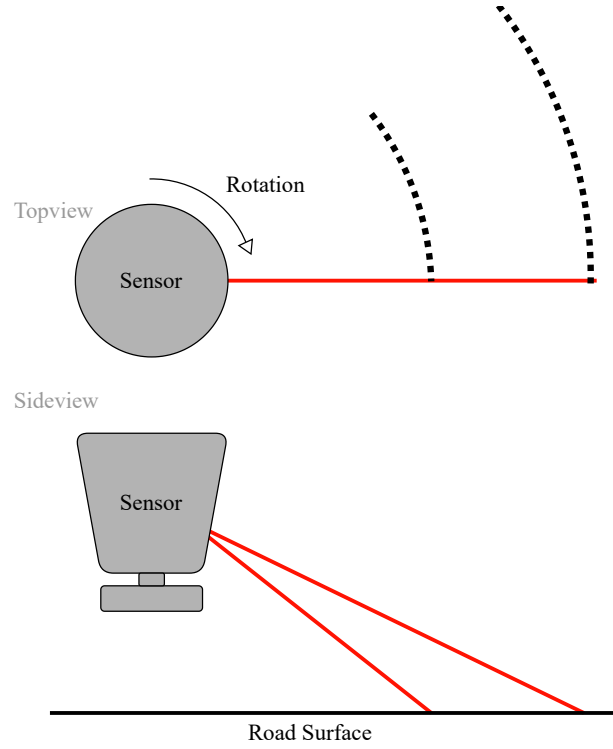


Figure 4.23: On a flat surface the rotating *LiDAR* creates a circular scan pattern on the road surface. In our evaluation, we fit a circle to triplets of points that lie on the road surface (using the semantic information provided by SemanticKITTI/CARLA), and estimate the noise on the range measurement by comparing the distance of a fourth point to the fitted circle.

the sensor w.r.t. the road, and can assume the road to be locally flat. As the *LiDAR* rotates at a constant speed, they create characteristic "rings" on the ground surface, see Fig. 4.23. We can compare noise along the ray w.r.t. the imaginary circle on a small patch of the road surface.

We use Scan Unfolding by Triess et al. [Tri+20] on the non-egomotion corrected point clouds from the SemanticKITTI dataset [Beh+19b] to obtain per-point projections into range images and semantic segmentation images, see Fig. 4.14. In sterile simulation data, the rings are perfectly circular, while in real data, they are not, but for most point clouds, this assumption holds at least locally, cf. Fig. 4.18.

In the segmentation/range image, we select all possible quadruplets of points that lie on the same layer (scanline) on the road surface. Assume we have four neighboring points that are all labeled as "road" (color ■ in Fig. 4.14): $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_n$, where $\mathbf{p}_i = (x_i, y_i, z_i)^T \in \mathbb{R}^3$. We then fit a circle (in the ground plane) to three of these points ($\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$), and calculate the euclidean error of the fourth point ($\mathbf{p}_n$) w.r.t. the fitted circle.

First, we select $\mathbf{p}_1$ to be the origin of the reference coordinate system. Then we calculate the new unit vectors $\mathbf{x}$ and $\mathbf{z}$:

$$\begin{aligned}
 \tilde{\mathbf{p}}_2 &= \mathbf{p}_2 - \mathbf{p}_1 \\
 \tilde{\mathbf{p}}_3 &= \mathbf{p}_3 - \mathbf{p}_1 \\
 \tilde{\mathbf{p}}_n &= \mathbf{p}_n - \mathbf{p}_1 \\
 \mathbf{x} &= \frac{\tilde{\mathbf{p}}_2}{\|\tilde{\mathbf{p}}_2\|} \\
 \mathbf{z} &= \frac{\mathbf{x} \times \tilde{\mathbf{p}}_3}{\|\mathbf{x} \times \tilde{\mathbf{p}}_3\|}
 \end{aligned} \tag{4.44}$$

Next, we calculate the unit vector $\mathbf{y}$:

$$\mathbf{y} = \mathbf{z} \times \mathbf{x}. \tag{4.45}$$

We then build a rotation transform $\mathbf{R}$:

$$\begin{aligned}
 \mathbf{R} &= \begin{bmatrix} \mathbf{x}^T \\ \mathbf{y}^T \\ \mathbf{z}^T \end{bmatrix} \\
 \mathbf{q}_1 &= \mathbf{R}\tilde{\mathbf{p}}_1 = \mathbf{0} \\
 \mathbf{q}_2 &= \mathbf{R}\tilde{\mathbf{p}}_2 \\
 \mathbf{q}_3 &= \mathbf{R}\tilde{\mathbf{p}}_3 \\
 \mathbf{q}_n &= \mathbf{R}\tilde{\mathbf{p}}_n
 \end{aligned} \tag{4.46}$$

By construction, $\mathbf{z}$ is orthogonal to both $\mathbf{x}$ and $\tilde{\mathbf{p}}_3$, and since $\tilde{\mathbf{p}}_2$ is aligned with $\mathbf{x}$, $\mathbf{q}_2^{(2)}$ and $\mathbf{q}_3^{(2)}$ are now equal to 0, i.e. the points $\mathbf{q}_2$ and $\mathbf{q}_3$ lie in the ground plane.

A circle with center x_c, y_c and radius r on a plane is described by the equation:

$$x_c^2 + y_c^2 = r^2 \tag{4.47}$$

or for a general point in 2D with coordinates a, b :

$$\begin{aligned}
 (a - x_c)^2 + (b - y_c)^2 &= r^2 \\
 2ax_c + 2by_c + (r^2 - x_c^2 - y_c^2) &= a^2 + b^2
 \end{aligned} \tag{4.48}$$

We can then write the equation for the circle in the rotated coordinate system with the points $\mathbf{q}_2$ and $\mathbf{q}_3$, and then assemble and solve a linear system of equations $\mathbf{M}\mathbf{s} = \mathbf{b}$:

$$\begin{aligned}
 \mathbf{s} &= \begin{bmatrix} 2x_c \\ 2y_c \\ x_c^2 + y_c^2 - r^2 \end{bmatrix} \\
 \mathbf{b} &= \begin{bmatrix} 0 \\ (\mathbf{q}_2^{(0)})^2 + (\mathbf{q}_2^{(1)})^2 \\ (\mathbf{q}_3^{(0)})^2 + (\mathbf{q}_3^{(1)})^2 \end{bmatrix}
 \end{aligned} \tag{4.49}$$

and

$$\mathbf{M} = \begin{bmatrix} 0 & 0 & 1 \\ \mathbf{q}_2^{(0)} & \mathbf{q}_2^{(1)} & 1 \\ \mathbf{q}_3^{(0)} & \mathbf{q}_3^{(1)} & 1 \end{bmatrix} \quad (4.50)$$

$$\mathbf{s} = \mathbf{M}^{-1} \mathbf{b}$$

And can then compute the total noise by combining the noise from the xy and z direction:

$$\begin{aligned} \epsilon_{xy}^2 &= \left(\sqrt{(\mathbf{q}_n^{(0)} - x_c)^2 + (\mathbf{q}_n^{(1)} - y_c)^2} - r \right)^2 \\ \epsilon_z^2 &= (\mathbf{q}_n^{(2)})^2 \\ \epsilon &= \sqrt{\epsilon_{xy}^2 + \epsilon_z^2} \end{aligned} \quad (4.51)$$

This is repeated for all permutations of the points in the quadruplet, and for each quadruplet of neighboring road points in the dataset. The resulting errors are then aggregated into a histogram and visualized as heatmap, see Fig. 4.24 for comparison between the different simulation methods and real data. We see that the Plain model does not match the real distribution at all, since the points lie more or less on perfect circles, as indicated by the yellow hotspot the bottom of the histogram. [BlenSor](#) is a bit better, but the histogram from the physical model and real data match the best. We attribute this to the fact that the physical model matches the actual, real-world noise distribution better than the mere assumption of Gaussian noise made by [BlenSor](#). For example, with [BlenSor](#), there is no influence of material parameters or geometry of the target to the accuracy of the range measurement, but this influence exists [BM18]. In contrast to [BlenSor](#), the physical model actually models the shape of each pulse and is able to introduce more realistic noise into the measurements, for example depending on target geometry or its other properties such as metallicness or roughness.

4 Physically-based *LiDAR* Simulation

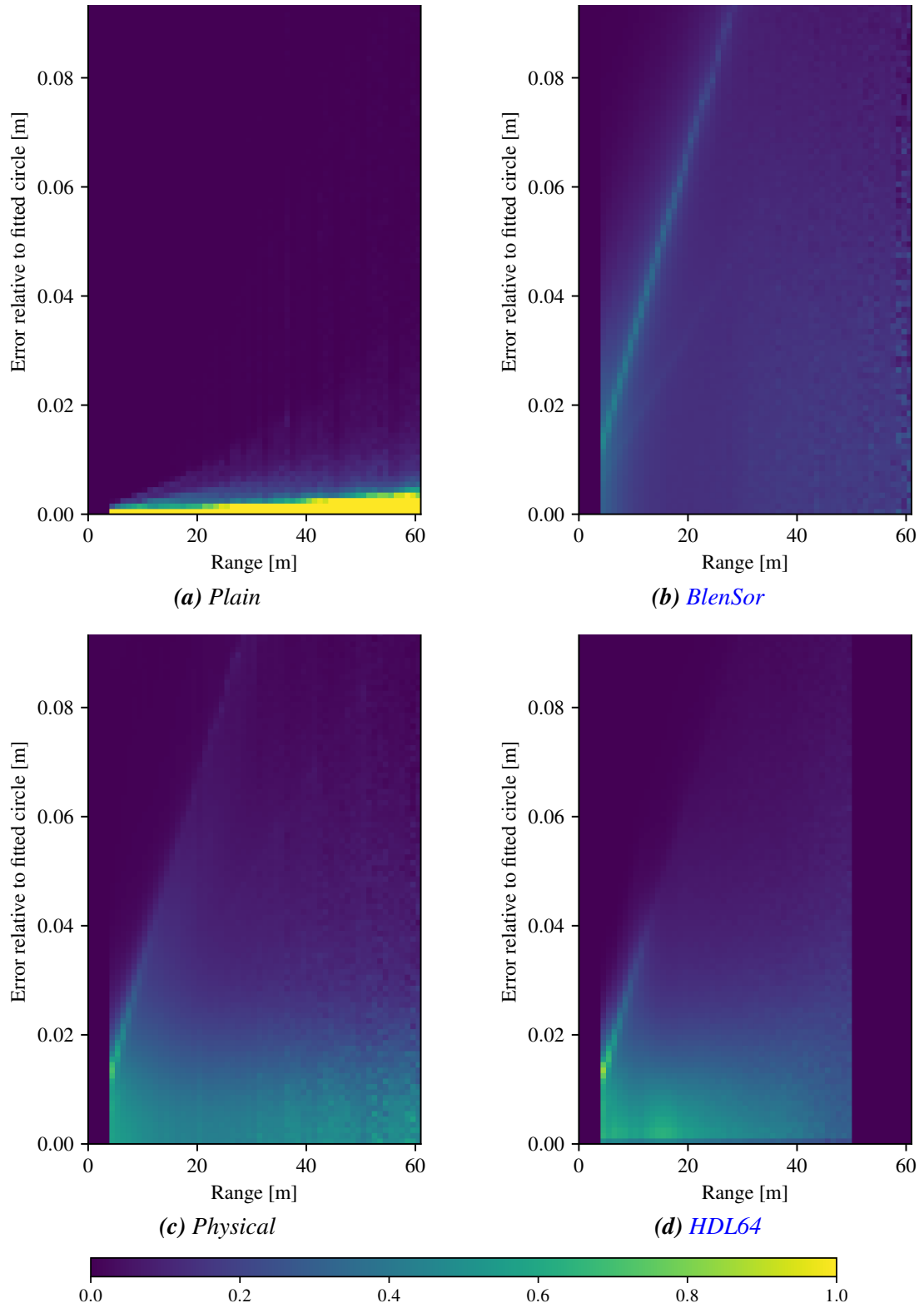


Figure 4.24: Comparison of the noise on the road surface for the different simulation models and the real KITTI data. Each histogram is normalized using its maximum value, and then mapped according to the colormap below.

4.6.4 Downstream Task: LiDAR Object Detection

From the early days of simulation, it has been observed that good performance of a model in simulation does not necessarily translate to good performance in the real world [Bro98; Her+01; Gai+16]. This phenomenon is often referred to as *domain gap*, i.e. the difference between the distribution of training data and evaluation data inhibits models to learn representations that are meaningful in both simulation and the real world, cf. Fig. 4.14. In the context of simulated and real data, this refers to the fact that each difference between the two data distributions (including artifacts which are just present in the simulated data) will be “exploited” by the model to solve a task, thereby hurting generalization to real-world data, where these artifacts are not present. The more realistic the simulated data is, the better the models trained on the simulated data should perform on real-world data, i.e. a smaller domain gap should lead to easier/better transfer of the model to real-world data. We therefore evaluate the simulated data by training three different state-of-the-art LiDAR object detection networks on the simulated data and evaluating them on real data, comparing the “performance” between simulation models.

Network Architectures

We evaluate the simulated data using three different LiDAR object detection networks:

PartA2 PartA2 by Shi et al. [Shi+21] is a two stage approach: The first stage of the network predicts 3D proposals by processing the point cloud using an Encoder-Decoder network. Each proposal is then divided into parts, which are scored and refined in the second stage: The second stage uses *Region of Interest (RoI)*-aware pooling to aggregate part information from each 3D proposal, scoring and refining boxes and their locations based on the part features from the first stage. This refinement stage is guided by the *IoU* between the predicted and ground truth boxes.

PointPillars PointPillars by Lang et al. [Lan+19] is a network that was developed specifically with efficiency and execution speed in mind. This is achieved by converting the point cloud into a 2D BEV representation that can be processed efficiently by a 2D CNN: Firstly, the raw point cloud is divided into a grid of vertical columns, called pillars. Then, for the points of each pillar, a shared Linear Layer is used to create a feature representation, and the result is scattered into a 2D BEV image. The BEV image is then processed by a 2D CNN Backbone, and its output is then in turn fed into a detection head. The anchor-based detection head predicts class scores, bounding box dimensions, orientation of potential objects in the scene.

PV-RCNN PV-RCNN by Shi et al. [Shi+20] is a two-stage 3D object detection network that combines a voxel-based 3D CNN and PointNet. The first stage uses a 3D voxel-based CNN for feature encoding and proposal generation. The second stage refines these proposals by using key points obtained via farthest-point-sampling from the original

Parameter	PartA2	PointPillars	PV-RCNN
Batch Size	4	4	2
Optimizer	Adam	Adam	Adam
Learning Rate	0.003	0.003	0.01

Table 4.3: The parameters we have chosen for training the networks in the simulation-to-real generalization experiment. Note: We do not deviate from the original training parameters of the respective networks as provided by OpenPCDet [Tea20].

point cloud in combination with RoI features. This combined approach improves on the limitations of previous voxel-based networks in regions of sparse data.

To reduce duplicates, all of the above networks use IoU-based Non-Maximum Suppression (NMS) to the predicted boxes.

Training Setup

We use OpenPCDet [Tea20] for training and evaluating the networks. The networks are trained according to Table 4.3. In experiments where we pre-train or fine-tune on real data we use the official training split from the KITTI Object dataset [Gei+13]. For the other experiments, where we pre-train on simulated data, the networks are pre-trained for 30k iterations on the simulated data from each of the simulation models. After 30k iterations of pre-training on simulated data, the model performance on the real data starts to degrade, so we stop the pre-training at this point.

Evaluation

We evaluate the networks on the official validation split of the KITTI Object dataset [Gei+13] and report the results in Table 4.4 under section “Pre-training”. Then, the networks are fine-tuned on either 100%, 10% or 1% of the real KITTI data, and once again evaluated on the same validation split. For each network, and each percentage of real data, we report the results for the different simulation models in Table 4.4 and Table 4.5 under section “Fine-tuning”. The results are reported in terms of the AP for the classes Car, Pedestrian and Cyclist, and mAP over all classes for both BEV and 3D.

In the case where no real data is used, two out of three networks achieve their best performance when pre-trained on the physical model (cf. Table 4.4). This indicates that the point clouds generated by our physical model match real-world point clouds more closely than the other simulators. However, when fine-tuning on 100% of the real data, all simulation models are outperformed by training on real data directly, as opposed to fine-tuning. At least for this experimental setup, all simulation models harm the performance of the networks compared to from-scratch training, i.e. the best simulator is the one that “poisons” the network the least, allowing the network to recover best when fine-tuning on real data. We attribute this to the fact that the domain gap between simulated and real world is too large regardless of the sensor simulation model used, and that 100% of

the real data is enough data to fully train the networks from scratch. When reducing the amount of real-world data to 10% or 1%, the pre-training shows more potential, with all simulation models outperforming the experiment without pre-training on simulated data. At 10% of the real data, using the Plain model as a simulator is the best choice, while at 1% of the real data, each of the simulation models works best with one of the networks. We suspect that the domain gap between simulated and the real world is so large, that the differences between how the different [LiDAR](#) simulation models depict the world becomes negligible in comparison to the general “world domain gap” of CARLA and the real world. In conclusion, when there is absolutely no annotated real-world data available, the physical simulator seems to be the best choice. However, as soon as there is at least some annotated data available, the advantage of the physical model is diminished. Still, simulated data is valuable for long-tail scenarios and scenarios that are too dangerous to collect real data for. Where possible, training on a sufficient amount of real data is always preferable.

		Pre-training						Fine-tuning on Full (100% of) Real Dataset																	
		Network		Car AP@IOU0.5	BEV	3D	Cycl. AP@IOU0.5	BEV	3D	Ped. AP@IOU0.5	BEV	3D	Overall mAP	BEV	3D	Car AP@IOU0.5	BEV	3D	Cycl. AP@IOU0.5	BEV	3D	Ped. AP@IOU0.5	BEV	3D	Overall mAP
Simulator	Network	Car AP@IOU0.5	BEV	3D	Cycl. AP@IOU0.5	BEV	3D	Ped. AP@IOU0.5	BEV	3D	Overall mAP	BEV	3D	Car AP@IOU0.5	BEV	3D	Cycl. AP@IOU0.5	BEV	3D	Ped. AP@IOU0.5	BEV	3D	Overall mAP	BEV	3D
None	Part A2	-	-	-	-	-	-	-	-	-	-	-	-	86.969	86.717	62.221	58.711	48.321	45.746	65.837	63.725				
	PointPillar	-	-	-	-	-	-	-	-	-	-	-	-	87.596	85.750	58.666	55.231	46.491	41.038	64.251	60.673				
	PV-RCNN	-	-	-	-	-	-	-	-	-	-	-	-	87.478	85.927	58.631	53.272	45.342	42.109	63.817	60.436				
Plain	Part A2	33.053	26.241	2.975	0.350	22.644	15.560	19.557	14.050	84.414	84.188	58.238	56.559	53.858	50.133	65.503	63.627								
	PointPillar	35.193	27.568	3.438	0.209	8.832	4.546	15.821	10.774	88.125	86.510	57.462	52.200	43.390	37.458	62.992	58.723								
	PV-RCNN	47.713	44.091	0.222	0.026	10.804	5.669	19.580	16.595	85.078	83.925	54.615	50.495	39.499	33.591	59.731	56.004								
Blensor	Part A2	27.214	21.668	0.000	0.000	14.342	10.225	13.852	10.631	84.364	82.732	56.696	52.451	45.527	41.298	62.196	58.827								
	PointPillar	36.484	32.389	1.871	0.067	13.549	7.010	17.301	13.155	86.648	85.390	50.232	47.582	42.835	37.721	59.905	56.898								
	PV-RCNN	51.967	50.061	0.097	0.000	7.081	4.195	19.715	18.085	85.086	84.882	53.511	48.637	41.380	37.670	59.992	51.898								
Physical	Part A2	25.530	23.411	3.702	1.068	17.939	12.452	15.724	12.310	84.859	84.260	59.513	56.031	48.520	42.924	64.297	61.072								
	PointPillar	53.759	47.484	1.204	0.143	8.762	3.580	21.242	17.069	89.586	87.886	53.189	50.662	45.265	39.817	62.680	59.455								
	PV-RCNN	48.972	47.040	0.120	0.012	15.993	9.519	21.695	18.857	87.004	86.807	48.356	46.204	40.910	36.973	58.756	56.661								

Table 4.4: Comparison of the different LiDAR simulators for object detection. None refers to the case where no pre-training was performed on any simulated data. For each network, the simulator where the network achieved the best overall performance is highlighted in bold and color-coded numbers. When only pre-training on simulated data is performed, our physical simulator outperforms the other simulators for two out of the three networks, indicating that the point clouds generated by our simulator more closely resemble real point clouds. However, when fine-tuning on 100% of the KITTI dataset, using no simulated data at all (None) performs best. All evaluations are conducted on the validation split of KITTI Object.

Simulator	Network	Fine-tuning on Reduced (10% of) Real Dataset						Fine-tuning on Reduced (1% of) Real Dataset								
		Car BEV	AP@IOU0.5 3D	Cycl. BEV	AP@IOU0.5 3D	Ped. BEV	AP@IOU0.5 3D	Overall mAP BEV	Car BEV	AP@IOU0.5 3D	Cycl. BEV	AP@IOU0.5 3D	Ped. BEV	AP@IOU0.5 3D	Overall mAP BEV	
None	Part A2	82.298	81.977	40.829	39.201	34.158	30.943	52.428	50.707	76.464	73.021	8.299	6.971	8.739	6.782	31.167
	PointPillar	84.469	82.400	54.200	47.195	32.567	26.528	57.079	52.041	81.793	78.234	12.373	10.336	18.989	11.725	37.718
	PV-RCNN	87.343	87.104	37.887	34.878	35.821	33.510	53.684	51.831	80.275	78.723	14.331	13.797	16.697	14.144	37.101
Plain	Part A2	83.336	82.929	50.471	46.206	51.302	46.139	61.703	58.425	77.873	75.703	13.380	11.724	16.158	13.168	35.804
	PointPillar	85.243	84.165	47.519	44.035	38.074	31.319	56.946	53.173	81.544	79.346	14.252	11.932	26.586	21.842	40.794
	PV-RCNN	84.469	82.953	50.989	47.086	38.642	34.743	58.033	54.927	81.635	79.894	12.210	10.467	30.135	25.458	41.327
Blensor	Part A2	81.385	79.645	45.182	42.218	47.751	42.899	58.106	54.921	74.003	72.014	7.306	5.191	27.102	21.403	36.137
	PointPillar	87.070	85.161	49.445	43.784	34.959	29.462	57.158	52.802	82.194	79.733	14.141	11.243	29.952	25.765	42.095
	PV-RCNN	88.221	86.536	44.931	40.792	37.628	32.774	56.926	53.368	77.686	75.994	15.285	12.720	23.715	19.282	38.895
Physical	Part A2	78.692	78.428	51.771	48.449	46.476	42.686	58.980	56.521	74.753	72.729	10.189	8.487	28.930	24.421	37.957
	PointPillar	83.339	81.420	44.104	39.015	36.237	31.599	54.560	50.678	82.993	80.865	12.992	10.473	24.412	20.258	40.133
	PV-RCNN	83.298	82.854	43.780	41.178	37.892	34.124	54.990	52.719	82.568	80.812	11.640	10.106	20.768	17.348	38.325
																36.089

Table 4.5: Comparison of the different *LiDAR* simulators for object detection using artificially reduced real-world data. None refers to the case where no pre-training was performed on any simulated data. Fine-tuning on 10% of KITTI, the Plain simulation model performs best, using no noise or visibility model. Fine-tuning on 1% of KITTI, it is a tie between the Plain, *Blensor* and our physical model, where each model performs best for one of the networks. All evaluations are conducted on the validation split of KITTI Object.

4.6.5 Measuring Suitability for Validation

Another possibility to evaluate the performance of a simulator, besides measuring the performance as a training data generator for downstream tasks, is to measure suitability to reliably evaluate real-world methods on simulated data. Using simulation to evaluate models on the long tail of the scenario distribution and on underrepresented safety critical scenarios provides a safe way to test and refine methods without the risk and cost associated with real-world testing. However, evaluating safety-critical methods on simulated data requires a simulator that is as close to reality as possible, i.e. methods should perform identically on simulated and real data and vice versa. This means, that a method trained on real data, with specific success and failure characteristics, should exhibit identical characteristics also on simulated data, otherwise the simulator is not suitable for validation. This can be measured using the “ground-truth detection agreement”-score introduced by Manivasagam et al. [Man+20]. For this evaluation, ground-truth labels have to be the same for both real and simulated data, i.e. the objects must be of the same class, at the same location, with the same size and orientation in simulation as they were annotated in the real world.

The agreement score is then calculated as:

$$\kappa_{det} = \frac{|G_+ \cap S_+| + |G_- \cap S_-|}{|G_+ \cup G_-|}. \quad (4.52)$$

G refers to the ground-truth labels in the real world data, that have been detected (G_+) or missed (G_-) by the model. S refers to the *same* ground-truth labels (but in the simulated data), that have been detected (S_+) or missed (S_-) by the model. Essentially, the labels are bit-identical for both domains, only the input data to the model is different. The more realistic a simulator is, the higher the agreement score will be. To this end, we generated paired samples of real (KITTI Object) and simulated data for the various simulators: We replicated all scenarios from the KITTI Object dataset (validation split) in the simulator and recorded simulated *LiDAR* data for these scenarios. The perfect simulator should generate data that is indistinguishable from real data. This would lead to identical performance of the model on both real and simulated data. This especially includes the model making the same mistakes on both data sources, leading to ground-truth detection agreement of $\kappa_{det} = 1$. If the performance on simulated data and real-world data would be completely uncorrelated, the agreement score would be $\kappa_{det} = 0$.

Table 4.6 reveals multiple interesting insights: The agreement scores on cars are generally higher than on pedestrians and cyclists, which is plausible, as cars are modeled much better than pedestrians and cyclists in the CARLA simulator. Furthermore, we see that using the Plain model as a simulator leads to the best overall performance. This is surprising, as visually (Section 4.6.2) and statistically (Section 4.6.3) the Plain model was resembling the real data the least. We can only speculate as to why this might be: One suspicion is that the simulation models that apply noise and visibility thresholds do so in a way that is slightly different from how real-world sensors function, which is essentially adding another level of complexity to the same simulated source data. A model trained on

LiDAR Simulation	Network	κ_{Car}		κ_{Cyclist}		$\kappa_{\text{Pedestrian}}$	
		@IOU0.5	@IOU0.7	@IOU0.25	@IOU0.5	@IOU0.25	@IOU0.5
Plain	Part A2	0.756	0.600	0.535	0.478	0.637	0.585
	PointPillar	0.788	0.577	0.553	0.501	0.581	0.466
	PV-RCNN	0.785	0.593	0.662	0.610	0.706	0.609
BlenSor	Part A2	0.737	0.592	0.482	0.440	0.630	0.584
	PointPillar	0.776	0.564	0.552	0.531	0.624	0.508
	PV-RCNN	0.770	0.585	0.625	0.557	0.665	0.588
Physical	Part A2	0.749	0.573	0.522	0.460	0.638	0.587
	PointPillar	0.768	0.547	0.525	0.494	0.579	0.449
	PV-RCNN	0.794	0.582	0.631	0.564	0.679	0.588

Table 4.6: Detection Agreement Score on KITTI for different networks and *LiDAR* simulations. The best score for each network is highlighted in bold, color coded for each of the networks. The IOU thresholds are measured for oriented BEV bounding boxes. The scores are much higher for the Car class compared to the Cyclist and Pedestrian classes. Generally speaking, the Plain model performs better than *BlenSor* and the physical model.

real-world, noisy data over the course of the training learns a representation of what a car, cyclist or pedestrian looks like. It could be, that the data generated by the Plain simulation is the closest to this representation, as the physical and *BlenSor* model introduce noise and visibility thresholds that are only similar to the real world, but not identical and thus the statistical distribution of the simulated data (at least from the networks' perspective) becomes even more different from the real data. However, this is speculation and further research is needed to confirm this hypothesis.

4.7 Conclusion

We conclude the chapter by discussing the limitations of our model, and the implications of our findings. Finally, we close with a summary of the chapter.

4.7.1 Assumptions and Resulting Limitations

Our model relies on several assumptions. These assumptions lead to some limitations, which are discussed in the following.

Flat, Homogeneous Target Surface: We assume that the target surface is flat and homogeneous in its material properties. This is a simplification, as real-world targets are often complex and have varying material properties. However, this simplification is necessary to make the model tractable. From a practical standpoint, the model can be extended to more complex modeling of targets by segmenting the target into smaller, homogeneous parts and applying the model to each part separately. However, as we extract data from the z-buffer of a 3D scene, we would require at least twice the vertical **and**

horizontal resolution of the depth map to capture the required level of detail, slowing down the simulation significantly. For close-range targets, where the laser beam is spread over a small area, this assumption is not critical: For our model, the beam width is approximately 0.04 m at 10 m distance, and 0.12 m in diameter at 50 m distance. For far-away targets, where the laser beam is spread over a larger area due to the beam divergence, this assumption prohibits our model from accurately capturing multi-material targets and multi-returns.

No Atmospheric Damping and Bad Weather: We neglect atmospheric damping, as it is insignificant compared to the $\frac{1}{R^2}$ term in (4.12). This assumption is valid for clear weather conditions, where the meteorological visibility is over 20 km. For a generally lowered visibility range, it is trivial to add the damping factor β to the model in (4.8). To model actual reflections caused by water droplets or snowflakes, a significant extension to CARLA would be required, as rain and snow are not simulated as true 3D objects in the environment at the moment. Furthermore, to evaluate this extension, a dataset with bad weather data would be required. As there is no bad weather data contained in the KITTI dataset, we did not investigate this further.

Emitter and Receiver Optics are Co-located: The optical center for the emitter and receiver optics are the same for our simulator. This limitation is a result of using the z-buffer as data source for our simulator, as the z-buffer only provides the range to the content of the pixel. To model separate emitter and receiver locations, a ray tracing approach would be required. This would allow us to model the path of the laser beam from the emitter to the target and back to the receiver, and to take shadow and occlusion effects into account. We tried to mitigate this limitation by using an empirical approach for shadowing effects in Section 4.3.6, and the results look visually promising. However, further research is needed to introduce a more physically accurate model via ray tracing for shadowing effects.

4.7.2 Summary

Using visual and statistical comparisons (cf. Section 4.6.2, Section 4.6.3, and Fig. 4.24), we demonstrated that the noise characteristics of our physical model align more closely with real-world data than those of our baselines. In terms of visibility characteristics, both our baseline BlenSor and our physical model exhibit a closer match to real-world characteristics compared to using no visibility model; however, based on visibility range alone, determining the best choice from our comparisons remains challenging. As a source of training data for 3D LiDAR object detectors when no annotated real-world data is available, our physical model appears to be the most suitable (cf. Table 4.4). However, once even a small amount of annotated real-world data becomes available, there is no longer a clear optimal choice among the simulation models. For evaluating the simulation models as tools for validating object detection networks, we operated under the assumption that if a LiDAR simulation model produces realistic measurements, perception models trained on real data should perform comparably on previously unseen simulated data. However, we found that increased realism (in terms of visual appearance

and noise characteristics) in simulated data does not necessarily translate to a more similar performance between simulated data and data from the real world.



(a) *Simplistic CARLA world.*



(b) *Karlsruhe, Germany.*

Figure 4.25: *CARLA world and Karlsruhe, Germany (KITTI dataset).*

In summary, we see that despite our best efforts to create a realistic, physically-based [LiDAR](#) simulator, the domain gap between simulated and real-world data remains significant. Surprisingly, this is true for all sensor simulation models in our experiments. Even our baseline model [BlenSor](#), with its simple noise and visibility model should be a strict improvement over the Plain simulation model, with its unrealistic perfectly planar surfaces, but our experiments revealed, that this is not necessarily the case. We believe that the main cause for the domain gap does not lie in “how” the simulated world is mapped to recorded sensor data (i.e. the modeling of the sensor itself), but rather the characteristics of the underlying simulated world, that deviate too much from our real-world target domain: CARLA is recorded in various simulated towns, which are modeled after small American country towns and small cities. These differ a lot from the “dense” European-style city of Karlsruhe, where the KITTI dataset has been recorded. At the same time, we know

from literature, that modern, state-of-the-art LiDAR object detectors are so sensitive to data distributions, that even just differing object sizes between Germany and the US [Wan+20b] lead to performance degradation, despite the data being recorded using the same sensor. We can assume the domain gap between simulated and real data to be much greater than between two real-world recordings (cf. Fig. 4.25), and as a result the model performance suffers more severely going from the simulated to the real-world domain, since state-of-the-art object detectors are capable to exploit any nuance or artifacts present in the data.

4.7.3 Outlook

One potential idea for future work could be to take a more data-driven simulation approach, at least for cases, where there is a real sensor model with sufficient pre-recorded data available. Recently, NeRF approaches have made big advances in the space of novel view synthesis for cameras [Mil+20]. Since for all real-world AD datasets, there is paired camera and LiDAR data available, one idea might be to extract the 3D mesh along with the material parameters from the real world using a NeRF-based approach, such as [Mun+22]. Since the LiDAR beam geometry, target surface geometry and the surface material parameters are then known, the simulated LiDAR model could be calibrated to match the real-world sensor data. This could be achieved by formulating the task as a reconstruction problem, i.e. given the real-world mesh, geometry and material parameters, reconstruct the real-world LiDAR measurements using the simulated LiDAR model, in essence calibrating the simulation LiDAR model to match the real-world sensor behavior. For inference, the calibrated simulation model could then be used to generate measurements from a simulated world, since in simulation we have access to the material parameters and the geometry of the target. Taking this approach still requires the simulated assets to have realistic material parameters, which could be solved by combining semantics (for example generated by an off-the-shelf semantic segmentation network) with the NeRF, to generate the material parameters for the simulated assets, by sampling the material parameters from a database of real-world materials.

5 Self-Supervised LiDAR Scene Flow and Motion Segmentation

This chapter (except the outlook in Section 5.5) has been published in S. A. Baur, D. J. Emmerichs, F. Moosmann, P. Pinggera, B. Ommer, and A. Geiger. “SLIM: Self-Supervised LiDAR Scene Flow and Motion Segmentation”. In: *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*. IEEE, 2021, pp. 13106–13116. DOI: [10.1109/ICCV48922.2021.01288](https://doi.org/10.1109/ICCV48922.2021.01288). URL: <https://doi.org/10.1109/ICCV48922.2021.01288> [Bau+21]. For detailed information on the contributions of the authors, please refer to Appendix B.

Recently, several frameworks for self-supervised learning of 3D scene flow on point clouds have emerged. Scene flow inherently separates every scene into multiple moving agents and a large class of points following a single rigid sensor motion. However, existing methods do not leverage this property of the data in their self-supervised training routines which could improve and stabilize flow predictions. Based on the discrepancy between a robust rigid ego-motion estimate and a raw flow prediction, we generate a self-supervised motion segmentation signal. The predicted motion segmentation, in turn, is used by our algorithm to attend to stationary points for aggregation of motion information in static parts of the scene. We learn our model end-to-end by backpropagating gradients through Kabsch’s algorithm and demonstrate that this leads to accurate ego-motion which in turn improves the scene flow estimate. Using our method, we show state-of-the-art results across multiple scene flow metrics for different real-world datasets, showcasing the robustness and generalizability of this approach. We further analyze the performance gain when performing joint motion segmentation and scene flow in an ablation study. We also present a novel network architecture for 3D LiDAR scene flow which is capable of handling an order of magnitude more points during training than previously possible.

5.1 Introduction

Reasoning about the motion of dynamic objects in a scene is crucial for robotic applications such as autonomous driving. One representation for such motion is scene flow, a set of displacement vectors between two consecutive time frames, forming a 3D vector field $\mathcal{F}_{t \rightarrow t+1}$. Its application allows to transform points $\mathcal{P}_t$ recorded in the first frame into the second frame via $\mathbf{p}_i^{(t+1)} = \mathbf{p}_i^{(t)} + \mathbf{f}_i$. Another representation is the binary classification of points into either *moving* or *stationary*, known as motion segmentation.

Our method tackles the problem of simultaneous LiDAR scene flow estimation and motion segmentation using a deep network. Existing works treat these two problems independently. Most of them rely on large amounts of labeled data to obtain state-of-the-art results (see Section 3.2.2). Self-supervised learning attempts to alleviate this problem, but existing scene flow networks using weak or self-supervision have critical drawbacks: Currently, all of them require downsampling of point clouds from around 128,000 points for typical LiDAR frames to no more than 8,192 points during training or they train and evaluate on datasets with 1-to-1 point correspondences, see Section 3.2.3.

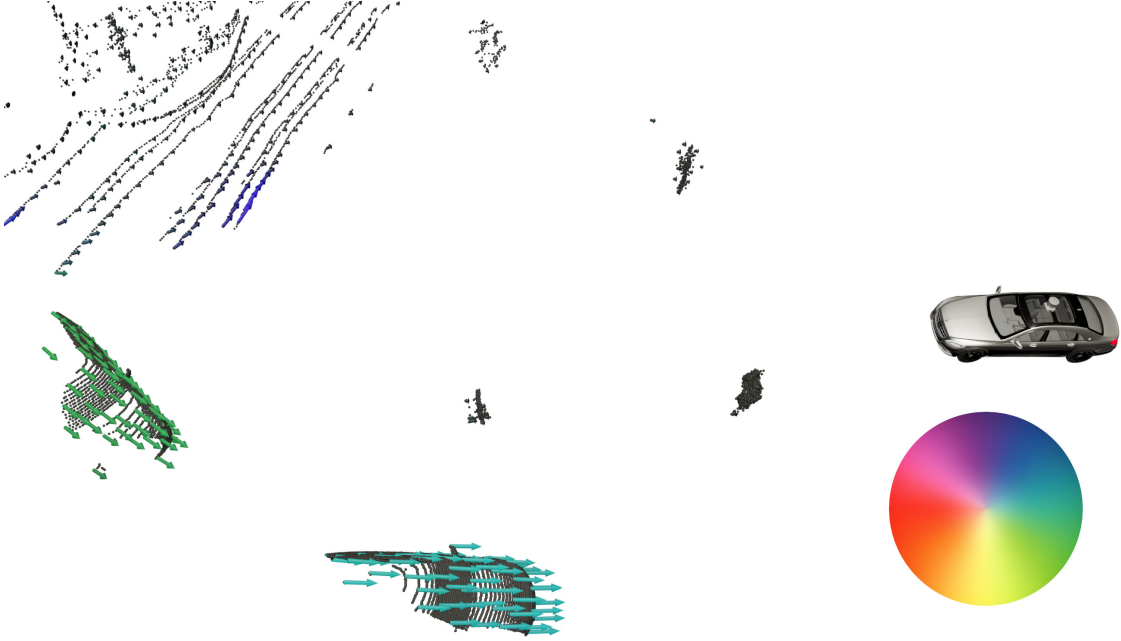


Figure 5.1: Scene Flow Prediction by SLIM. Flow vectors are colored by their direction and magnitude as illustrated with the color wheel. Our method correctly annotates the background as static and produces consistent results for the two vehicles, matching the ground truth scene flow.

Our approach trains well even on datasets with no 1-to-1 correspondences and can handle one to two orders of magnitude more points per point cloud than previous approaches for self-supervised scene flow. We build on top of RAFT [TD20], the state-of-the-art for optical flow, and extend the network to perform iterative scene flow estimation and motion segmentation on PointPillar-based BEV feature representations of point clouds [Lan+19]. This architecture realizes an excellent trade-off between generalizability, accuracy, and computational efforts. Using the consistency of motion for the stationary background, we are able to extract training targets for self-supervised motion segmentation. No additional data or labels besides raw, consecutively recorded point clouds are required for training, the cues for self-supervised classification come from the typical motion profiles of these classes alone.

Our main **contributions** are:

- Our method is the first point-cloud-based scene flow estimation method that can classify points as "moving" and "stationary" based on self-supervised training.
- Our method significantly outperforms previous methods in point-cloud-based scene flow estimation, especially in terms of generalization to previously unseen data, which we demonstrate on multiple datasets, in both the self-supervised and fully supervised setting.
- Our novel network architecture can handle significantly more points than current weakly or self-supervised approaches.

5.2 Method

3D scene flow estimation uses two consecutive input point clouds $\mathcal{P}_t \in \mathbb{R}^{N \times 3}$, $\mathcal{P}_{t+1} \in \mathbb{R}^{M \times 3}$ to predict for each point in the first point cloud a 3D displacement vector, representing the motion of each point w.r.t. the sensor frame.

5.2.1 Network Architecture

We make use of an efficient network, specialized on **LiDAR** data from the perspective of an **Autonomous Vehicle (AV)**, sacrificing the ability to generalize to more freely moving scenes and objects as seen in FT3D for example. Our network consists of three components, detailed in Fig. 5.2: A point cloud encoder, a flow backbone, and an output decoder explained below. We also introduce the used loss functions and their purposes.

Point Cloud Encoder: The input point clouds $\mathcal{P}_t, \mathcal{P}_{t+1}$ are separately encoded into a **BEV** pseudo-image using the **Pillar Feature Net (PFN)** introduced in [Lan+19] (shared weights), the resulting values $\mathcal{I}_t, \mathcal{I}_{t+1} \in \mathbb{R}^{H \times W \times C}$ are then processed by the backbone. To ensure comparability with other methods [LQG19; Gu+19; Wu+20; Tis+20; Wei+20; Goj+21], we use for all datasets the same **BEV** extent covering the square from $-35m \leq x, y \leq 35m$ around the ego vehicle, with x, y being the horizontal axes. We use a resolution of $H = W = 640$ which corresponds to a pillar size of roughly 11cm.

Flow Backbone: Our backbone is heavily inspired by RAFT [TD20] which was developed to predict dense optical flow on images. Its core component is an update block acting recursively on a hidden state and a flow prediction, producing a more refined and accurate flow with every iteration. Therefore, a correlation volume is constructed from independently encoded input images, using the previous flow prediction to look up correlation values and thus guide the flow to a more accurately matching pixel region. Even though RAFT was designed for dense optical flow, we show that it is also very suitable and generalizes well in the sparsely populated **BEV** domain, which consists of more scattered and smaller regions and very independent motion patterns (moving traffic participants) than regular images.

We adapt RAFT [TD20] to handle flow prediction and iteratively update two additional logits, as shown in Fig. 5.2. The first logit map $\mathcal{L}_{cls}$ is used as an output signal to classify

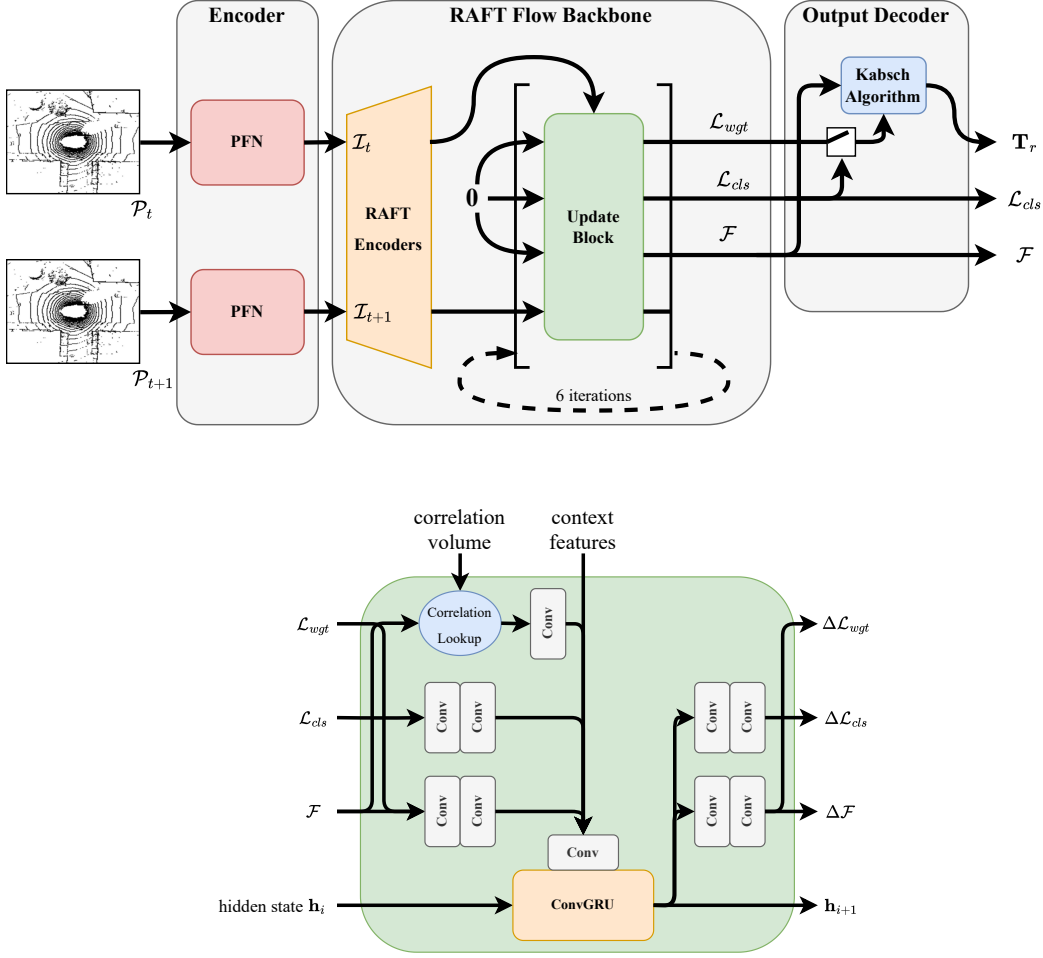


Figure 5.2: SLIM Network overview (top): Two point clouds are encoded into a *BEV* pseudo-image and processed by the RAFT Flow Backbone. The output decoder aggregates the flow using Kabsch’s algorithm and the point-wise classification logits. **SLIM Update Block (bottom):** The Convolutional Gated Recurrent Unit (ConvGRU) iteratively predicts updates for the flow and logits, using correlation lookups guided by the predicted flow.

the points as stationary or moving w.r.t. the world frame. The accuracy of predicted flow can vary greatly in a scene, as featureless surfaces are not suitable for flow estimation. The second logit $\mathcal{L}_{wgt}$ is used to overcome this problem, allowing the network to signal its confidence in a flow estimate. Both logits are used by the output decoder to aggregate and improve accuracy on static and dynamic scene elements.

$\mathcal{L}_{cls}$ is handled similarly to the flow stream, but the task of confidence weighting is more closely tied to the flow prediction and thus the data streams are therefore coupled during the processing of information. Except for this minor change we retain the general framework of RAFT, including zeroing the gradient not only on the input flow but also on the input logits of each update block.

Output Decoder: Our backbone produces a 2D *BEV* flow map $\mathcal{F} \in \mathbb{R}^{H \times W \times 2}$ together

with two logit maps, $\mathcal{L}_{cls}, \mathcal{L}_{wgt}$. Additionally, our model keeps track of a global classification threshold p_{stat} through moving averages, discussed in more detail in Section 5.2.2.

Firstly, the output decoder uses these BEV maps to annotate each point of the input point cloud $\mathcal{P}_t$ with a flow vector $\mathbf{f}_i$ and two logits $l_{cls,i}, l_{wgt,i}$ according to the values of its corresponding pillar cell. By doing this we assume that all points in a pillar behave very similar. We believe this to be true for almost all LiDAR point clouds measured outdoors, as all moving traffic participants need to occupy some ground. Additionally, most LiDAR systems are built in a way that their beams are not upward-facing. Note that although our network architecture is specialized in this way, our loss framework is suited for any 3D scene flow prediction and does not require the assumption of a 2D flow. In order to regularize and improve the flow prediction on the static scene, the output decoder aggregates the points classified as still into a single coherent rigid-motion odometry transform $\mathbf{T}_r \in \mathbb{R}^{4 \times 4}$. This aggregated transform should be minimizing the squared error when comparing its resulting flow with the flow prediction from the network:

$$\mathbf{T}_r = \operatorname{argmin}_{\mathbf{T} \in \mathbb{R}^{4 \times 4}} \sum_{\mathbf{p}_i \in \mathcal{P}_t} w_i \|(\mathbf{T} - \mathbf{I}_4)\mathbf{p}_i - \mathbf{f}_i\|^2 \quad (5.1)$$

Defining the odometry transform $\mathbf{T}_{\text{ego}}^{t \rightarrow t+1} \in \mathbb{R}^{4 \times 4}$ as the ego vehicle position at time $t + 1$ measured in the ego vehicle frame from time step t , we know that all stationary points should have a flow of $((\mathbf{T}_{\text{ego}}^{t \rightarrow t+1})^{-1} - \mathbf{I}_4)\mathbf{p}_i$. We therefore expect the computed transform $\mathbf{T}_r^{-1} \approx \mathbf{T}_{\text{ego}}^{t \rightarrow t+1}$ to be a good approximation for the odometry of the vehicle.

We use Kabsch’s algorithm [Kab76] to compute $\mathbf{T}_r$ using singular value decomposition which is differentiable. The weights w_i for each point determine how much influence each flow vector prediction has on the final result $\mathbf{T}_r$. We first apply a sigmoid activation to the confidence logits and then mask them based on the classification logits. Afterwards, we normalize all weights to sum to 1 to ensure numerical stability. See Section 2.3.1 for more details on the computation of the rigid transform.

$$w_i = \frac{\sigma(l_{wgt,i})[\sigma(l_{cls,i}) \geq p_{stat}]}{\sum_j \sigma(l_{wgt,j})[\sigma(l_{cls,j}) \geq p_{stat}]} \quad (5.2)$$

The confidence logits only receive gradient updates through the computation of $\mathbf{T}_r$ and are therefore trained end-to-end without further supervision.

For the purpose of inference, the final aggregated flow map of our network is

$$\mathbf{f}_{agg,i} = \begin{cases} (\mathbf{T}_r - \mathbf{I}_4)\mathbf{p}_i & \text{if } l_{cls,i} \geq p_{stat} \\ \mathbf{f}_i & \text{if } l_{cls,i} < p_{stat} \end{cases} \quad (5.3)$$

or in words: Points predicted to be stationary use the flow from the aggregated rigid ego-motion transform, all other points use the raw flow prediction from the network. However, we make use of the aggregated transform $\mathbf{T}_r$ and the classification logits $\mathcal{L}_{cls}$ in the loss. Especially $\mathcal{L}_{cls}$ does not have any gradients yet through the aggregated flow or transform and is only trained through special loss terms introduced in the next section.

5.2.2 Loss

We have different self-supervised components based on the two input point clouds, and the network’s predicted flow, logits, and the aggregated transform.

K-Nearest Neighbors (k-NN) Loss: First and foremost we use a **k-NN** loss as a self-supervision signal for our flow estimation as is done by earlier works for self-supervised scene flow [MOH20; Wu+20; Tis+20]. Given a pointwise flow $\mathbf{f}_i$ and a nearest neighbor function $NN_{\mathcal{P}_{t+1}}(\mathbf{p}_i) = \operatorname{argmin}_{\mathbf{p}_j \in \mathcal{P}_{t+1}} |\mathbf{p}_j - \mathbf{p}_i|$ we find for every flow-corrected point in $\mathcal{P}_t$ an NN-based error distance:

$$e_i = |NN_{\mathcal{P}_{t+1}}(\mathbf{p}_i + \mathbf{f}_i) - (\mathbf{p}_i + \mathbf{f}_i)| \quad (5.4)$$

As we work with noisy data and especially non-bijective point clouds there are greater errors observed in distant points. To mitigate this, we define an outlier percentage $p_{out} \ll 1$ and ignore all errors lying in this top-percentile. Additionally, we apply this loss not to the aggregated flow $\mathcal{F}_{agg}$ but to the raw flow prediction $\mathcal{F}$ and the rigid flow $\mathbf{f}_{r,i} = (\mathbf{T}_r - \mathbf{I}_4)\mathbf{p}_i$ separately, resulting in a total NN loss

$$L_{nn} = \frac{1}{|\mathcal{P}_t|} \sum_i e_i(\mathbf{f}_i) + e_i(\mathbf{f}_{r,i}) \quad . \quad (5.5)$$

We found that this stabilizes the training in the early phases and does not hurt the performance. Applying the loss only to the aggregated flow has the problem that early during the training the smooth and small rigid flow is more accurate and thus dominates over the raw flow leading the network into a local optimum where the classification predicts only stationary points. This then leads to the raw flow getting masked out from the aggregated flow, consequently receiving no gradient updates, and thus performing badly. Of course, depending on training settings and network initialization, this behavior could also occur in reverse. Thus we train both flow fields every time at every point.

Rigid cycle consistency: Inspired by recent successful applications of cycle consistency losses [Tis+20; MOH20] we apply our network not only on the input point clouds $(\mathcal{P}_t, \mathcal{P}_{t+1})$ but also on the reversed order $(\mathcal{P}_{t+1}, \mathcal{P}_t)$, giving us access to the prediction of the inverse rigid motion T'_r . For accurate predictions, these two rigid transforms are the exact inverses to each other. In order to measure the error, we apply them to the input point cloud as a way of weighting the different error aspects of the transforms.

$$L_{rcc} = \frac{1}{|\mathcal{P}_t|} \sum_{\mathbf{p}_i \in \mathcal{P}_t} |(\mathbf{T}'_r \mathbf{T}_r - \mathbf{I}_4)\mathbf{p}_i| \quad (5.6)$$

Artificial Labels: The classification logits $\mathcal{L}_{cls}$ are trained using a self-supervised loss based on the NN errors e_i . Additionally, we compute these errors not only for the raw flow prediction $\mathbf{f}_i$ but also for the rigid counterpart $\mathbf{f}_{r,i}$ in order to compare those $e_{r,i}$ to the errors for the raw flow. We use the standard binary cross-entropy loss based on artificial

labels depending on which NN error is smaller.

$$L_{al} = - \sum_{\mathbf{p}_i \in \mathcal{P}_t} [e_i < e_{r,i}] \log \sigma(l_{cls,i}) + [e_i \geq e_{r,i}] \log(1 - \sigma(l_{cls,i})) \quad (5.7)$$

Supervised loss: On datasets where ground truth flow labels $\mathcal{F}_{gt}$ and vehicle odometry $\mathbf{T}_{ego}^{t \rightarrow t+1}$ are available, it is possible to train the network using supervised losses:

$$L_{flow} = \frac{1}{|\mathcal{P}_t|} \sum_{\mathbf{p}_i \in \mathcal{P}_t} |\mathbf{f}_{gt,i} - \mathbf{f}_i| \quad (5.8)$$

Additionally, we supervise the classification logits by specifying a ground truth rigid class target through a threshold m_{thresh} to the non-rigid flow part

$$r_i = \left[|\mathbf{f}_{gt,i} - ((\mathbf{T}_{ego}^{t \rightarrow t+1})^{-1} - \mathbf{I}_4) \mathbf{p}_i| \leq m_{thresh} \right]. \quad (5.9)$$

The resulting cross-entropy-based classification loss is

$$L_{cls} = - \frac{1}{|\mathcal{P}_t|} \sum_{\mathbf{p}_i \in \mathcal{P}_t} r_i \log \sigma(l_{cls,i}) + (1 - r_i) \log(1 - \sigma(l_{cls,i})). \quad (5.10)$$

The last supervised loss is used to increase the performance of the rigid transform mostly by providing a signal to the confidence weights used by the computation of the transform.

$$L_r = \frac{1}{|\mathcal{P}_t|} \sum_{\mathbf{p}_i \in \mathcal{P}_t} |(\mathbf{T}_r - (\mathbf{T}_{ego}^{t \rightarrow t+1})^{-1}) \mathbf{p}_i| \quad (5.11)$$

Classification Threshold: We observe in our experiments that our scores and metrics depend on a well-calibrated threshold p_{stat} for switching between the case of moving or stationary points. As our training pipeline already requires this threshold for the weights in the computation of $\mathbf{T}_r$ we require an online mechanism to adjust this threshold, instead of choosing it during post-processing. Therefore, we track moving averages of the metric $m(p)$ we want to optimize for a fixed set of possible thresholds $\{p_0, p_1, \dots, p_N\}$. We can then define the threshold $p_{stat} = p_i$ with

$$i = \underset{i=0, \dots, N}{\operatorname{argmin}} m(p_i) \quad (5.12)$$

as the point where our metric is currently optimal. It is important to note, that in the case of self-supervised trainings we only make use of the [k-NN](#) errors e_i and $e_{r,i}$ which do not require any ground truth labels but also only approximate the metric we want to optimize. In the supervised setting, on the other hand, we use the ground truth flow to compute the moving averages of the metric as accurately as possible. For all our experiments we optimize the threshold according to the AEE 50-50 metric which we introduce in the following section. Please refer to Appendix C for more details.

5.3 Evaluation

To verify our claims, we first report results for different experimental setups and compare our method against two other baselines using in total 4 different datasets. In a final set of experiments, we highlight the importance of the different components of our method through an ablation study.

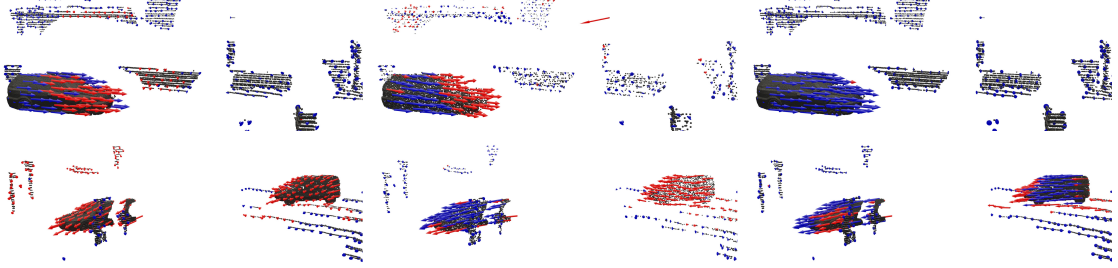


Figure 5.3: *Qualitative comparison of methods on a KITTI-SF scene. Accurately estimated flow according to AccR is colored blue, inaccurate predictions red. From left to right: PPWC, PF, Ours*

5.3.1 Experimental Setup

We applied only minor modifications to the [PointPillars \(PP\)](#) PFN encoder and the RAFT backbone (called RAFT-S in [TD20]). Details on our architecture can be found in Appendix C. We trained all our experiments for 100k iterations with a batch size of 1 and a learning rate of 10^{-4} using the RMSProp optimizer. In order to stay consistent with previous evaluation protocols [MOH20; Tis+20; Wu+20] we also remove all points up to 30cm above the ground from the point cloud prior to training and inference.

When training self-supervised we set the weights of the supervised losses to 0 and weighted the rest of the losses with $\lambda_{nn} = 2, \lambda_{rcc} = 1, \lambda_{al} = 0.1$. In the case of supervised trainings we only used the supervised losses with $\lambda_{flow} = 10, \lambda_{cls} = 1, \lambda_r = 10$.

Evaluation Metrics

To evaluate the performance, we use the established metrics and an extra outlier metric:

- **AEE:** average endpoint error (EE) across all points
- **AccS:** point ratio where $EE < 0.05$ m or relative error < 0.05
- **AccR:** point ratio where $EE < 0.1$ m or relative error < 0.1
- **Outl:** point ratio where $EE > 0.3$ m or relative error > 0.1
- **ROutl:** point ratio where $EE > 0.3$ m and relative error > 0.3

To ensure comparability with previous works we report Outl as a metric on KITTI-SF, however, we found it has several drawbacks, the most notable being that often the sum of Outl and AccR is larger than 100%, which is counterintuitive for an outlier metric. On the other datasets, we exclusively report the robust variant ROutl defined above. A more detailed discussion and direct comparison of these two metrics can be found in Appendix C.

As the other datasets we investigate contain a larger imbalance of moving scene parts w.r.t. stationary ones, we report scores separately for those two classes. Ground truth odometry is used to compute the non-rigid flow component

$$\mathbf{f}_{nr,gt,i} = \mathbf{f}_{gt,i} - ((\mathbf{T}_{ego}^{t \rightarrow t+1})^{-1} - \mathbf{I}_4)\mathbf{p}_i. \quad (5.13)$$

Points with a non-rigid flow larger than $m_{thresh} = 0.05$ m (corresponds to 1.8 km h^{-1}) are labeled as dynamic, the rest as stationary. We introduce AEE 50-50 as the average between the AEE measured separately on stationary and on moving points.

Baselines

We selected two methods as baselines which provide recent state-of-the-art results for self- as well as supervised training of scene flow: The first baseline we compare ourselves against, PointPWC (PPWC) [Wu+20], uses a backbone similar to RAFT [TD20] in that it also relies on correlation volumes inside its architecture. The second baseline is taken from [Tis+20] and learns to predict the ego-motion component of the flow separately from the non-rigid flow, similar to our approach in the way that we also consider the ego-motion as a special case in our architecture. We refer to this method as PF in the following.

Datasets

To obtain representative results, we use three very different outdoor traffic scene datasets which only have in common that they observe the scene through a LiDAR from the position of an (autonomous) vehicle taking part in the traffic. Additionally, we use FT3D [May+16] for the baselines as training data.

FlyingThings3D: Both baseline methods make extensive use of the FT3D [May+16] dataset, consisting of randomly flying objects. The point clouds and the flow are 1-to-1 corresponding and have significantly more points than their counterparts from the other datasets. This dataset is not suited for our method as we make the assumptions that the flow is mainly in the horizontal x, y -plane and that points inside the same pillar have a similar movement pattern. Both of these assumptions do not hold for this dataset, therefore we do not use it to train our method.

nuScenes: nuScenes [Cae+20] is an AD dataset recorded using a Velodyne VLP32 LiDAR sensor, featuring densely annotated object boxes in space and time, as well as a pointwise semantic annotation. We annotated the point clouds with ground truth information using the recorded vehicle odometry for static points and the tracking information (bounding boxes) of moving objects.

CARLA Simulator: Using the CARLA Simulator [Dos+17b], we collected an artificial dataset with 142k consecutive point clouds with ground truth flow annotation. Town07 is used exclusively for evaluation, where we recorded 1,900 different scenes.

KITTI: The KITTI-SF [MHG18] training dataset is used by many works as the main benchmark for 3D scene flow [LQG19; Wu+20; Tis+20; KER20]. For these 142 samples, stereo ground truth and flow annotations are given, allowing conversion of the dataset to a point cloud representation with scene flow labels. In contrast to the nuScenes and CARLA dataset, it consists of pairs of 1-to-1 corresponding point clouds and flow, which differs from real-world point clouds measured at different time steps. We also compare our method against the baselines on this dataset. For training, we use the whole collection of unlabeled raw *LiDAR* point clouds published with the KITTI [Gei+13] dataset (abbreviated with KITTI-RL), which have no annotations.

5.3.2 Impact of Point Cloud Density

Our network architecture is able to process a full scene from KITTI-RL as input, comprising roughly 64,000 points. While in theory the baselines can run inference on a similar size, during training it is infeasible to run them with more than the proposed 8,192 points as the memory requirements and the training time increase dramatically.

Therefore, increased point cloud density does not only provide more information but also represents a domain shift for the networks during inference. Our first set of experiments shows which type of evaluation produces better results for the baselines and our method and determines the evaluation mode we use in the following experiments.

Table 5.1 shows the results of these experiments. We train the baselines in a self-supervised fashion on FT3D, as it gives them the best scores when evaluating on KITTI-SF. For our network, we train self-supervised on KITTI-RL, as it, in turn, produces the best result on KITTI-SF.

We conclude that both baselines suffer from the domain shift due to different point densities, and do not benefit from the increased resolution. Therefore, we evaluate the baselines always on the downsampled point clouds and our method on the full point clouds.

5.3.3 Training Dataset and Domain Shift

The baselines have been evaluated on KITTI-SF with the networks trained on FT3D. We find that our method performs quite differently on KITTI-SF based on which dataset is used during training. Therefore, Table 5.2 showcases the validation performance of self-supervised models on KITTI-SF for different training datasets.

Our method gives the best results for KITTI-RL. This makes sense because except for the small camera *FOV* in KITTI-SF the geometric properties and their distribution match. The baselines give better results when trained on FT3D than on any other dataset. However, out of the other three *LiDAR* datasets, KITTI-RL seems to be the best choice.

	Eval Data	#points	AEE↓	AccS↑	AccR↑	Outl↓
PPWC [Wu+20]	FT3D	8192	0.1610	0.2035	0.5809	0.7811
		65536	0.1654	0.1999	0.5783	0.7876
	KITTI-SF	8192	0.3466	0.0820	0.3145	0.8233
		all	0.3648	0.0726	0.2974	0.8579
PF [Tis+20]	FT3D	8192	0.1726	0.0601	0.2336	0.9832
		65536	0.1755	0.0584	0.2126	0.9832
	KITTI-SF	8192	0.3009	0.0986	0.2296	0.9738
		all	0.3256	0.1104	0.2058	0.9778
Ours*	KITTI-SF	8192	0.1207	0.5178	0.7956	0.4024
		all	0.0668	0.7695	0.9342	0.2488

Table 5.1: Influence of point cloud density. Our method (*) is trained on KITTI-RL.

FT3D working better than KITTI-RL is also not surprising, as it resembles more the camera FOV as found in KITTI-SF. Qualitative results are shown in Fig. 5.3.

5.3.4 Evaluation on LiDAR Data

For an evaluation closer to the real-world we perform self-supervised experiments on the two annotated datasets without point correspondences, CARLA, and nuScenes. Both datasets exhibit a much larger imbalance of moving points (non-rigid flow $> m_{thresh}$) compared to static points. Where in KITTI-SF around 40% of points are moving, only $\approx 5\%$ move in nuScenes and CARLA. We, therefore, compute the AEE and Acc scores separately and report also the newly introduced AEE 50-50 for easier comparison of methods. Table 5.3 shows the results for the nuScenes dataset and Table 5.4 the results for CARLA, where we trained our method as well as the baselines on the respective datasets.

As one can see, both baseline methods perform much worse than our method. However, comparing their results to the ones they achieved on KITTI-SF suggests that the methods themselves have no real problem with missing correspondences when trained on more realistic data.

5.3.5 Supervised Flow Prediction

Finally, we present the best results for each method on the different datasets. All results are achieved using supervised training except in the case of our method and the KITTI-SF dataset. Here, our method performs best when trained self-supervised on KITTI-RL compared to supervised training on any of the other two annotated datasets, demonstrating the capabilities of our self-supervised model.

Table 5.5 and Table 5.6 show that our method greatly outperforms the baselines w.r.t. almost all metrics. The domain gap between KITTI-RL and KITTI-SF is the smallest of

	Train Data	AEE↓	AccS↑	AccR↑	Outl↓
PPWC [Wu+20]	FT3D	0.3569	0.0870	0.3202	0.8204
	nuScenes	0.7708	0.0897	0.2078	0.9193
	CARLA	0.8131	0.0094	0.0663	0.9928
	KITTI-RL	0.3712	0.1992	0.4092	0.7406
PF [Tis+20]	FT3D	0.3009	0.0986	0.2296	0.9738
	nuScenes	0.4628	0.0000	0.1356	1.0000
	CARLA	0.3955	0.1586	0.3866	0.9565
	KITTI-RL	0.4448	0.0526	0.2310	0.9885
Ours	nuScenes	0.1013	0.7156	0.8739	0.3046
	CARLA	0.1454	0.5106	0.7899	0.3900
	KITTI-RL	0.0668	0.7695	0.9342	0.2488

Table 5.2: Self-supervised training with domain shift, evaluation on KITTI-SF

	Moving			Stat.	50-50
	AEE↓	AccR↑	ROutl↓	AEE↓	AEE↓
Zero	0.6381	0.1632	0.5783	0.5248	0.5814
PPWC [Wu+20]	0.3539	0.2543	0.3848	0.1974	0.2756
PF [Tis+20]	0.7399	0.0000	0.9364	0.0570	0.3985
Ours	0.1050	0.7365	0.0240	0.0925	0.0987

Table 5.3: Self-supervised training & evaluation on nuScenes

	Moving			Stat.	50-50
	AEE↓	AccR↑	ROutl↓	AEE↓	AEE↓
Zero	0.4049	0.1805	0.4753	0.4752	0.4401
PPWC [Wu+20]	0.3811	0.0948	0.5057	0.2577	0.3194
PF [Tis+20]	0.5711	0.0085	0.9153	0.1093	0.3402
Ours	0.0809	0.8351	0.0528	0.0853	0.0831

Table 5.4: Self-supervised training & evaluation on CARLA

Method	Train Data	AEE↓	AccS↑	AccR↑	Outl↓
PPWC [Wu+20]	FT3D	0.2578	0.1803	0.4838	0.7176
PF [Tis+20]	FT3D	0.1743	0.2716	0.5830	0.8729
Ours	KITTI-RL	0.0668	0.7695	0.9342	0.2488

Table 5.5: Best results on KITTI-SF for all methods. Note that PPWC and PF are trained using full supervision while our method uses only self-supervision.

	Method	Moving			Stat.	50-50
		AEE↓	AccR↑	ROutl↓	AEE↓	AEE↓
CARLA	PPWC [Wu+20]	0.2205	0.2456	0.4257	0.2120	0.2163
	PF [Tis+20]	0.8143	0.0000	0.9024	0.1357	0.4750
	Ours	0.04311	0.9547	0.0043	0.0466	0.0449
nuScenes	PPWC [Wu+20]	0.2341	0.4949	0.2034	0.0845	0.1593
	PF [Tis+20]	0.7934	0.0000	1.0000	0.0415	0.4175
	Ours	0.0702	0.8921	0.0170	0.0499	0.0600

Table 5.6: Best supervised results

all, thus our network achieves the best result on KITTI-SF using self-supervision. The other methods are not able to profit from the small domain gap to the same extent.

In case of nuScenes and CARLA the baselines struggle with the imbalance of moving points. Especially PF achieves promising results for the stationary points even outperforming our method on nuScenes. However, it completely ignores the moving objects. In contrast, our method uses a moving classification threshold and therefore gracefully adapts to datasets with changing amounts of moving points. We conclude that our method robustly handles different datasets with a consistent flow AEE 50-50 of roughly 5cm whereas the baselines perform drastically different and worse on the more realistic datasets.

5.3.6 Ablation Study

We now perform a series of experiments to ablate the importance of individual components, starting with our most basic setup: A PP encoder [Lan+19] with a RAFT backbone as a strong baseline, see Table 5.7. We first modify the RAFT backbone to update and track the flow map together with classification logits, which we use to apply an aggregation step to the masked ego-motion flow map in order to regularize the motion field for improved accuracy on the static parts of the scene. Then, another map of logits for producing weights used by the aggregation step is introduced, in theory, allowing the network to select regions with high flow certainty and resulting in even stronger performance on static scene elements. For the supervised training we observe a steady improvement in performance. However, in the self-supervised case, we observe that the confidence weights do not yield further improvements compared to pure aggregation. We attribute this to the

	logit introd.	ego aggr.	conf. weights	Mov. AEE	Stat. AEE	50-50 AEE
Self-Super- vised	✓	✓		0.1214 0.1050	0.1889 0.0925	0.1551 0.0987
	✓	✓	✓	0.1413	0.1100	0.1256
Super- vised				0.1273 0.0933	0.1530 0.0474	0.1401 0.0704
	✓	✓	✓	0.0702	0.0499	0.0600

Table 5.7: Ablation study on nuScenes

fact that during self-supervised training, the moving thresholds are updated only through the very noisy *k*-NN residuals. This is insufficient to guide the moving threshold to an optimal calibration. For a more detailed analysis, we refer to Appendix C.

5.3.7 Motion Segmentation

As demonstrated in Table 5.7, the motion segmentation improves scene flow performance. Qualitative analysis of the logits in Fig. 5.4 reveals that the network indeed learns to pick out moving objects from the point clouds. With self-supervised training on KITTI-RL, our model achieves a *Mean Intersection over Union (mIoU)* score of 59.5% with a sensitivity of 73.1% on KITTI-SF. A more detailed discussion of motion segmentation can be found in Appendix C.

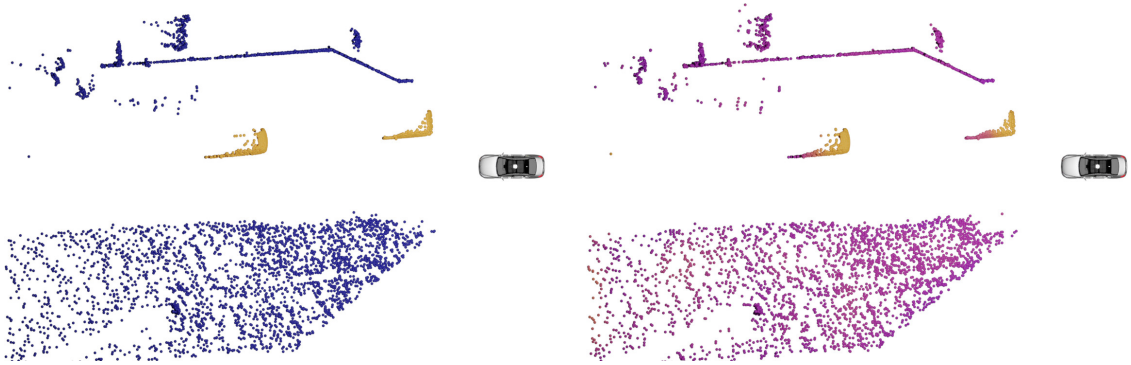


Figure 5.4: Left: Ground truth motion segmentation, right: Predicted dynamicness, higher moving probability brighter

For more in-depth evaluations the reader is referred to the appendix in Appendix C.

5.4 Conclusion

We introduced SLIM, a new method for **LiDAR** scene flow estimation which improves over the state-of-the-art through several key contributions: We presented a new network design, allowing us to train and evaluate on larger point clouds, hence removing the need for downsampling. Our model classifies stationary and moving parts of the scene dynamically. We demonstrated that the resulting aggregation of static flow leads to a significant improvement of the rigid flow field, in turn enabling better self-supervised classification into stationary and moving points. In future work, we plan to explore better regularization strategies and more self-supervisory signals to further robustify our method.

5.5 Outlook

The advancements in this chapter provide a solid foundation for further developments in **LiDAR** scene flow estimation. Additionally, the accurate **LiDAR** scene flow generated by our method can be transformed into something even more useful for **AD**: The motion information contained in accurate scene flow offers a valuable basis for subsequent self-supervised tasks.

For this reason, in the next chapter, we use the **LiDAR** scene flow estimated by our method to tackle the challenge of self-supervised **LiDAR** object detection. We demonstrate how clustering the **LiDAR** scene flow estimated by our method allows us to identify an initial set of *moving* objects. Through **LiDAR** scene flow-based tracking, we refine these clusters by discarding implausible tracks and enhancing the plausible ones. We employ an iterative training framework which generalizes to *movable* objects, incorporating plausible tracks into the training data as they are discovered. This enables us to train **LiDAR** object detectors from scratch without relying on any human annotations.

Thus, the contributions of this chapter are significant on their own, but also serve as a step towards achieving state-of-the-art **LiDAR** object detection performance without using human annotations, as we will demonstrate in the next chapter.

6 LiDAR-only Self-Supervised 3D Object Detection

This chapter has been published in S. A. Baur, F. Moosmann, and A. Geiger. “LISO: Lidar-Only Self-supervised 3D Object Detection”. In: *Computer Vision - ECCV 2024 - 18th European Conference, Milan, Italy, September 29-October 4, 2024, Proceedings, Part LXXXVI*. Ed. by A. Leonardis, E. Ricci, S. Roth, O. Russakovsky, T. Sattler, and G. Varol. Vol. 15144. Lecture Notes in Computer Science. Springer, 2024, pp. 253–270. DOI: [10.1007/978-3-031-73016-0_15](https://doi.org/10.1007/978-3-031-73016-0_15). URL: https://doi.org/10.1007/978-3-031-73016-0_5C_15 [BMG24]. For detailed information on the individual contributions of the authors, please refer to [Appendix B](#).

3D object detection is one of the most important components in any Self-Driving stack, but current state-of-the-art LiDAR object detectors require costly and slow manual annotation of 3D bounding boxes to perform well. Recently, several methods emerged to generate pseudo ground truth without human supervision, however, all of these methods have various drawbacks: Some methods require sensor rigs with full camera coverage and accurate calibration, partly supplemented by an auxiliary optical flow engine. Others require expensive high-precision localization to find objects that disappeared over multiple drives.

We introduce a novel self-supervised method to train state-of-the-art LiDAR object detection networks, requiring only unlabeled sequences of lidar point clouds. We call this trajectory-regularized self-training. It utilizes a state-of-the-art self-supervised LiDAR scene flow network under the hood to generate, track, and iteratively refine pseudo ground truth. We demonstrate the effectiveness of our approach for multiple state-of-the-art object detection networks across multiple real-world datasets.

6.1 Introduction

Human developmental research reveals that infants less than one year of age are able to categorize animate and inanimate objects based on observed motion cues and can generalize this categorization to previously unseen objects [TPP14]. Yet, lidar object detectors with state-of-the-art performance are trained using manually selected and categorized annotations. These annotations are very expensive to obtain and become outdated quickly: new lidar sensors are coming to market regularly, and trained state-of-the-art object detectors are sensitive to changes of sensor characteristics, mounting position and other domain gaps [REG19; Xu+21; Thé+22; Che+23; Esk24; Woz+24], even varying

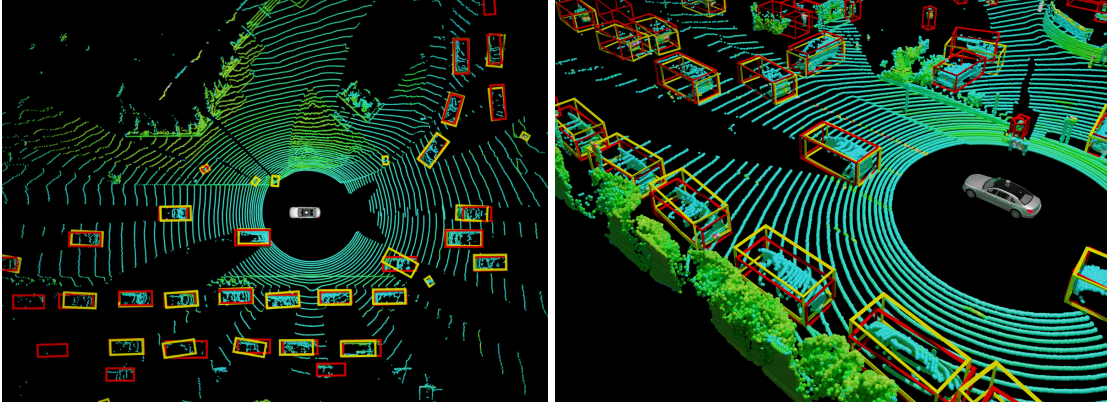


Figure 6.1: Objects predicted by our method using no manual annotations. *Red* boxes are ground truth boxes, *yellow* boxes are predicted by our network.

object sizes across different countries [Wan+20b], making existing annotations difficult to transfer, resulting in high re-labeling efforts with each change.

In this chapter, we aim at bridging this gap by distilling motion cues observed in self-supervised LiDAR scene flow into state-of-the-art single-frame LiDAR object detectors. We introduce a novel and robust self-supervised, easy-to-use method, operating solely on lidar point cloud sequences for easy-to-obtain 360° annotations. No human-annotated bounding boxes, cameras, costly high-precision GPS, or tedious sensor rig calibration is required, which could be especially useful in robotics research where data is often limited (even KITTI [GLU12b] only has forward facing cameras) or maintaining a multi-sensor-setup including its calibration is too cumbersome and expensive. Our method trains and runs a self-supervised LiDAR scene flow estimator [Bau+21] under the hood in order to create motion cues. We show in our experiments that these motion cues are a key factor to the success of our method. Based on the estimated LiDAR scene flow we bootstrap initial pseudo ground truth using simple clustering and track optimization. With these mined bounding boxes of *moving* objects we initialize a self-supervised, trajectory-regularized variant of self-training [Ami+23] (which is semi-supervised in its original form): We train a first version of a state-of-the-art object detector, then iteratively re-generate and trajectory-regularize the pseudo ground truth and retrain the detector. Since the single-frame object detector has no concept of motion, it generalizes to detect any *movable* object on the way. Exemplary output of the trained detector, 3D boxes in single-frame point clouds, is depicted in Fig. 6.1.

Our contribution is a novel self-supervised trajectory-regularized self-training framework for single-frame 3D object detection with the following properties:

- It is based entirely on LiDAR, i.e. without the limitations of prior works: no cameras, no calibration, no high-precision GPS, no manual annotations.
- It is agnostic w.r.t. the sensor model, mounting pose, and detector architecture and works with the same set of hyper parameters: We demonstrate this across four different datasets and different state-of-the-art detector networks.

- It is able to generalize from *moving* objects (motion cues) to *movable* objects (final detection results) and significantly outperforms state-of-the-art methods while being simpler.

We show that using motion cues together with trajectory-regularized self-training is key to this success. The code of this approach is [published](#) for easy use and comparison by other researchers.

In the following, the proposed method is described in detail in Section 6.2. Extensive experiments in Section 6.3 show the performance of our method before we conclude in Section 6.4.

6.2 Method

A general overview of our method is sketched in Fig. 6.2 and some steps are illustrated in Fig. 6.3. As input, we take raw (unlabeled) point cloud sequences and undergo three stages, all of which are detailed in the following: Preprocessing of point clouds and LiDAR scene flow computation (Section 6.2.1), initial pseudo ground truth generation (Section 6.2.2) and repeated training with pseudo ground truth refinement (Section 6.2.3). The final output of the method is a trained object detector which can detect *movable* objects in raw single-frame point clouds.

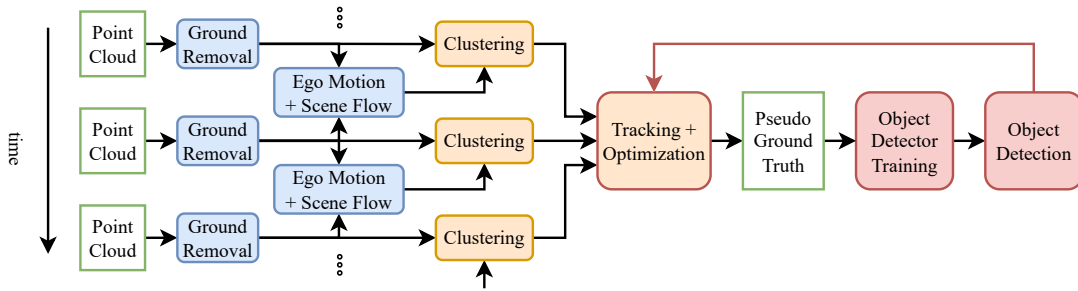


Figure 6.2: Overview of the proposed method. Point cloud sequences are preprocessed (blue, Section 6.2.1), initial pseudo ground truth is created (orange, Section 6.2.2) and the object detector is iteratively trained and pseudo ground truth regenerated (red, Section 6.2.3).

6.2.1 Preprocessing

We preprocess the raw input point clouds as follows:

Ground Removal: First, we remove distracting ground points from each single point cloud using JCP [She+21], which is a simple, robust, yet effective algorithm to remove ground points using changes in observed height above ground.

Ego Motion Estimation: Second, we compute ego-motion between neighboring frame pairs using KISS-ICP [Viz+23], which is based on a robust version of ICP. The output is

a cm-level accurate transformation $\mathbf{T}_{\text{ego}}^{t \rightarrow t+1} \in \mathbb{R}^{4 \times 4}$, describing the ego vehicle position at time $t + 1$ represented in the ego frame from time t .

LiDAR scene flow Estimation: Third, we compute LiDAR scene flow between neighboring frame pairs resulting in a flow vector $\mathbf{f}_i = (dx, dy, dz)^T$ for every point i in the first point cloud $\mathcal{P}^t$. We use SLIM [Bau+21] (cf. Chapter 5) as it is easy to use, features fast inference, and produces state-of-the-art results. The network is trained self-supervised on raw point cloud sequences, minimizing a k-nearest-neighbor loss between forward and time-reversed point clouds.

The components of our preprocessing steps (Ground Removal, Ego Motion Estimation, LiDAR scene flow Estimation) have been selected for robustness and are all used with their default parameters from their respective publications.

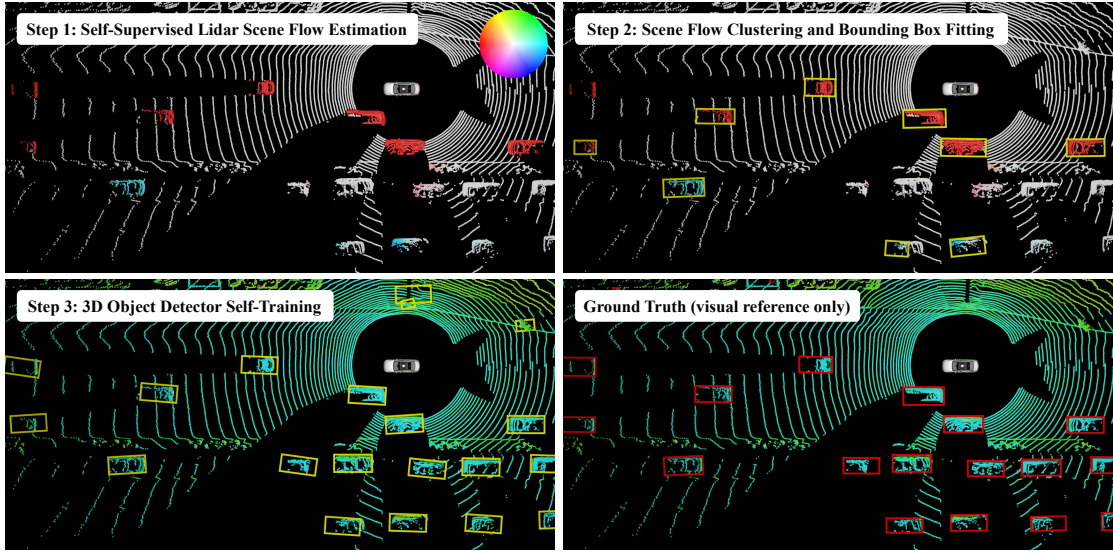


Figure 6.3: Overview over preprocessing, initial pseudo ground truth generation and training with examples. *Top left:* In the first step, self-supervised LiDAR scene flow is computed and corrected for vehicle ego-motion. Points are colored by flow direction and magnitude. *Top right:* In the second step, the scene flow is clustered and bounding boxes are fitted (to the moving objects), tracked and optimized. *Bottom left:* In the third step, the network is trained on the pseudo ground truth and is generalizing to static objects also, since it does not have the motion information as input signal. Points are thus colored by laser intensity. *Bottom right:* Ground truth, for reference.

6.2.2 Initial Pseudo Ground Truth Generation

The aim of our method is to mine pseudo ground truth for training a 3D LiDAR object detector. For a single point cloud $\mathcal{P}^t$ (consisting of $n \in \mathbb{N}$ points $\mathbf{p}_i$, $\mathbf{p}_i \in \mathcal{P}^t$) this ground truth is a set of 3D bounding boxes $\mathcal{B}$ with confidences representing the objects at time t :

$$\mathcal{B}^t = \{\mathcal{B}_j^t, j \in \mathbb{N}\} = \{(x, y, z, l, w, h, \theta, c)_j\} \quad (6.1)$$

Here, $(x, y, z) = \mathbf{x}$ define the center position, (l, w, h) the length, width and height, θ the heading (orientation around up axis), and c the confidence for a single box.

The key success factor of our method is to focus on a high precision (and potentially low recall) of the initial set of bounding boxes in order to avoid "wrong" objects to negatively influence the object detector. We achieve this by leveraging shape, density, and especially motion cues to robustly identify *moving* objects solely (see also top-right of Fig. 6.3). Section 6.2.3 targets at generalizing these to *movable* objects later on.

Flow Clustering: The points $\mathbf{p}_i \in \mathcal{P}^t$ in each preprocessed point cloud are clustered based on geometry and motion: All stationary points in a scene should have a flow $\mathbf{f}_i$ similar to the vehicle's ego-motion, i.e. $\mathbf{f}_i \approx \mathbf{f}_{i,\text{sta}} = ((\mathbf{T}_{\text{ego}}^{t \rightarrow t+1})^{-1} - \mathbf{I}_4) \cdot \mathbf{p}_i$. The residual flow must then be caused by motion of other actors: $\mathbf{f}_{i,\text{dyn}} = \mathbf{f}_i - \mathbf{f}_{i,\text{sta}}$. We filter all static points by applying a threshold of 1 m s^{-1} to the residual flow and cluster the remaining points based on their point location and flow vector in 6D using DBSCAN [Est+96](with parameters $\varepsilon = 1.0$, $\text{minPts} = 5$).

We fit a 3D bounding box $\mathbf{B}_j^t$ to each resulting cluster following [You+22] and discard boxes with $l/w > 4.0$, $lw < 0.35 \text{ m}^2$, or $lwh < 0.5 \text{ m}^3$. While these size-related thresholds may seem limiting, in Appendix D we demonstrate in an experiment that omitting these thresholds reduces the overall performance only slightly. The heading θ is set to match the "forward-axis" of the motion, i.e. we orient the boxes along the direction of the residual flow $\mathbf{f}_{i,\text{dyn}}$. The confidence c is initialized to 1 during clustering, and later replaced by detectors' confidences cf. Section 6.2.3.

Tracking: We run a simple flow-based tracker: Since we have accurate ego-motion available, we can track accurately in a fixed coordinate system w.r.t. the world. First, using the residual flow, we can compute a rigid body transform for each box at t that transforms the proposed box $\mathbf{B}_i^t$ forward in time towards its suspected location at $t + 1$, $\hat{\mathbf{B}}_i^{t+1}$. The propagated boxes $\hat{\mathbf{B}}^{t+1}$ are matched greedily against the actual boxes $\mathbf{B}^{t+1}$ based on box-center distance to the new detections. Unmatched boxes in $\mathbf{B}^{t+1}$ which are further away than 1.5 m from propagated boxes spawn new tracklets. Unmatched tracklets from $\mathbf{B}^t$ are propagated according to the last observed box motion for up to one time step, after that unmatched tracklets are terminated. We run tracking forward and reverse in time, connecting tracklets from forward and reverse tracking to tracks.

The resulting set of tracks is post-filtered: We discard tracks that are shorter than 4 time steps or that have a median box confidence c lower than the threshold 0.3. This retains only stable and high-confident tracks and avoids false positives to enter the pseudo ground truth.

Track Optimization: We reduce positional noise of the tracks by minimizing translational jerk on all tracks longer than 3 m. Let $\mathcal{X}_{\text{obs}}$ be the sequence of (noisy) observed box center positions $\mathbf{x}_i$ for consecutive time steps $i \in \{1, \dots, T\}$ of a track and their derivative $\frac{d\mathbf{x}_i}{dt} \approx \frac{\mathbf{x}_{i+1} - \mathbf{x}_i}{\Delta t}$. We compute smoothed track positions $\mathcal{X}_{\text{smooth}}$ by initializing them to $\mathcal{X}_{\text{obs}}$

and minimizing the following loss w.r.t. $\mathbf{x}_{\text{smooth}}$:

$$L = \sum_{i=1}^T \left\| \frac{d^3 \mathbf{x}_{i,\text{smooth}}}{dt^3} \right\|_2^2 + \beta \left\| \mathbf{x}_{i,\text{smooth}} - \mathbf{x}_{i,\text{obs}} \right\|_2^2 \quad (6.2)$$

I.e. we minimize the jerk $\frac{d^3 \mathbf{x}}{dt^3}$ and use a quadratic regularizer term ($\beta = 3$) on the positions. This simple optimization goal is better suited for our method than more sophisticated ones like an unrolled bicycle model, as it is both faster and also class-agnostic, i.e. more universal: Using a bicycle model is problematic when optimizing pedestrian tracks, since our method does not allow to distinguish different classes.

We subsequently align the orientation θ of each detection in a track to the direction of the smoothed track at its position. Box dimensions $\{l, w, h\}$ of all boxes in a track are adapted to the 90 percentile of observed box dimensions in a track. All hyperparameters related to clustering, tracking and track optimization have been tuned visually on two sample snippets from nuScenes [Cae+20].

6.2.3 Trajectory-Regularized Self-Training

After having mined initial pseudo ground truth of *moving* objects with a high precision and low recall, we now aim at iteratively improving our pseudo ground truth by training and using an object detector so that the pseudo ground truth generalizes to *movable* objects. We achieve this by executing iterative trajectory-regularized self-training, which is composed of the following two steps:

Training: We train the target object detection network using the current pseudo ground truth in a supervised training setup. Any single-frame object detection network can be plugged into our pipeline. In our experiments we do not deviate from the basic training setup of our object detection networks. Like any SOTA object detection method, we apply standard augmentation techniques to a point cloud during network training: Random rotation, scaling ($\pm 5\%$), and random translation up to 5 m around the origin. Furthermore, we randomly pick 1 to 15 objects from the pseudo ground truth database and insert a random subset of their points at random locations into the scene.

Pseudo Ground Truth Regeneration: We halt training after a set number of training steps s ($s = 30k$ in our experiments) and create new, improved pseudo ground truth by running the trained detector in inference mode over all sequences in the training dataset, using the detectors' confidence as box confidence c . We regularize these detections by running the flow-based tracker and track smoothing exactly like we do for the initial pseudo ground truth generation. Every 2nd iteration (i.e. every 60k steps in the experiments), we discard the network weights after regenerating the pseudo ground truth.

Fig. 6.4 illustrates the effect of our self-training: We see that the pseudo ground truth improves with each re-generation and that dropping network weights allows the network to re-focus on the generalized pseudo ground truth. Our strict requirements towards boxes being added to the pseudo ground truth leads to lower recall compared to the network, which as a result scores higher overall.

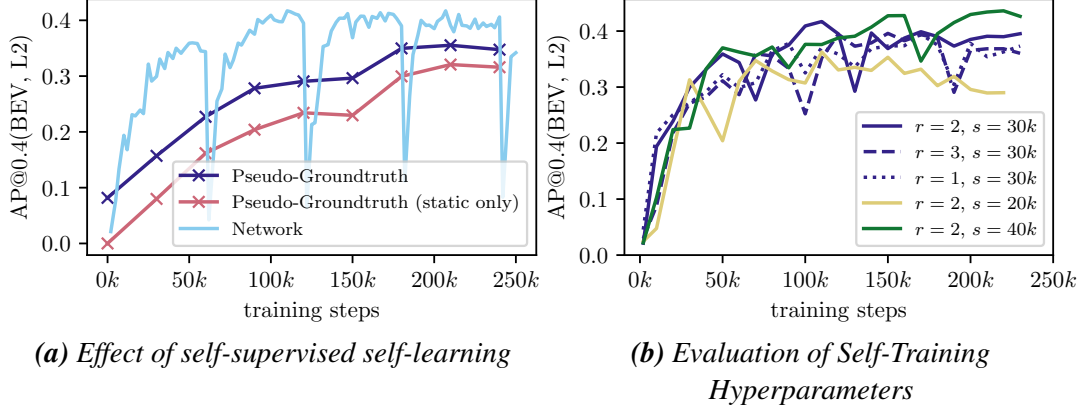


Figure 6.4: Left: Effect of self-supervised self-learning: At the beginning of the training, only moving objects are contained within the pseudo ground truth, which thus scores 0 on static objects. Thanks to the self-training, the network generalizes to movable objects and the score of the pseudo ground truth on static objects starts to increase with every regeneration at each "X" in the graph. The pseudo ground truth's performance is measured on a subset of the train set of WOD dataset since our tracking is looking ahead in time, which would be unfair w.r.t. other methods. The network (here: Centerpoint) is evaluated on a small subset of the validation set. **Right: Evaluation of Self-Training Hyperparameters:** Network (Centerpoint) performance over the course of a training on WOD. s is the number of training steps between pseudo ground truth regenerations, r is the number of regenerations after which network weights are dropped and re-initialized.

Two aspects are key to making our method perform well:

- Being restrictive when composing the pseudo ground truth (i.e. using plausible tracks of high-confidence network detections or clusters with significant flow only) avoids adding false positives into the pseudo ground truth and hence avoids that our network increasingly hallucinates with each iteration.
- Not using flow but only single-frame point clouds as input for the detector allows the detector to focus on appearance of objects solely.

These allow our method to generalize from initially mined *moving* objects to *movable* objects in the scene.

6.3 Evaluation

We evaluate our method on multiple datasets across multiple state-of-the-art networks. SLIM [Bau+21], the self-supervised flow network we use, is trained and inferred as in the published version, but BEV range is extended from $70 \times 70\text{m}$, 640×640 pixels to $120 \times 120\text{m}$ and 920×920 pixels.

		AV2				nuScenes	
		BEV-IOU		3D-IOU		AP $\uparrow$	AOE $\downarrow$
		AP@0.3 $\uparrow$	AP@0.5 $\uparrow$	AP@0.3 $\uparrow$	AP@0.5 $\uparrow$		
Sup.	CP [YZK21]	0.778	0.778	0.736	0.517	0.484	0.560
	TF [Bai+22]	0.783	0.654	0.767	0.601	0.627	0.501
Unsupervised	DBSCAN [Est+96]	0.054	0.026	0.020	0.003	0.008	0.171
	DBSCAN(SF)	0.070	0.026	0.065	0.015	0.003	0.082
	DBSCAN(GF)	0.105	0.041	0.088	0.024	0.000	1.000
	RSF [DZ23]	0.074	0.026	0.055	0.014	0.019	1.003
Self Train	Oyster [Zha+23] $\dagger$	0.354	0.245	-	-	-	-
	Oyster-CP [Zha+23]	0.381	0.266	0.150	0.002	0.091	1.514
	Oyster-TF [Zha+23]	0.340	0.198	0.182	0.016	0.093	1.564
	LISO-CP	0.448	0.335	0.367	0.188	0.109	1.062
	LISO-TF	0.417	0.294	0.317	0.176	0.134	0.938

Table 6.1: Evaluation on AV2 [Wil+21] and nuScenes [Cae+20]: CP, TF: network architecture, in the first two rows trained supervised for comparison. $\dagger$: Metrics as reported in [Zha+23]. SF: self-supervised *LiDAR* scene flow by SLIM, GF: ground truth *LiDAR* scene flow. Note that nuScenes uses a minimum precision and recall threshold of 0.1, and since the recall of GT flow clustering is lower than 0.1, all results are clipped away. For full nuScenes scores see Appendix D.

6.3.1 Datasets and Metrics

We evaluate our method on four different *AD* datasets. For a fair comparison, we compute metrics for *movable* objects by mapping all animate objects (Cars, Trucks, Trailers, Motorcycles, Cyclists, Pedestrians and other Vehicles) to a single category and discard all inanimate objects (Barrier, Traffic Cone,...) since our method does not predict any class attributes. In Table 6.2 and Table 6.3 we also give class-based results for completeness. Class labels for true positives are taken from ground truth. For false positives, the predicted class label is assigned randomly according to the label distribution in the dataset.

Waymo Open Dataset (WOD): [Sun+20] is a large, geographically diverse dataset recorded with a proprietary high quality lidar. We evaluate using the protocol of [Naj+22; Sei+24], using an area of whole 100m $\times$ 40m *BEV* grid around the ego vehicle, artificially cropping the predictions of our method down to this reduced area.

KITTI: [GLU12b; Gei+13] is recorded using a Velodyne HDL64 *LiDAR* sensor, where some parts have been annotated with 3D object boxes and tracking information in the forward facing camera field of view. A large portion of the published data is only available in raw format (KITTI-RL) without annotations, which our method is able to leverage, due to not requiring any ground truth annotations for training.

Argoverse 2: AV2 [Wil+21] is recorded using two stacked Velodyne VLP32 *LiDAR* sen-

		AP@0.4 ↑									AP@0.5 ↑		AP@0.7 ↑	
		Movable		Moving		Still		Vehi.	Ped.	Cycl.	Movable		Movable	
		BEV	3D	BEV	3D	BEV	3D	BEV	BEV	BEV	BEV	3D	BEV	3D
Sup.	CP [YZK21]	0.765	0.684	0.721	0.624	0.735	0.657	0.912	0.513	0.134	0.719	0.578	0.510	0.202
	TF [Bai+22]	0.746	0.723	0.714	0.668	0.733	0.710	0.918	0.457	0.216	0.711	0.669	0.532	0.385
Unsupervised	DBSCAN [Est+96]	0.027	0.008	0.009	0.000	0.027	0.006	0.184	0.002	0.001	0.019	0.002	0.000	0.000
	DBSCAN(SF)	0.026	0.010	0.064	0.041	0.000	0.000	0.073	0.010	0.009	0.011	0.001	0.000	0.000
	DBSCAN(GF)	0.114	0.071	0.318	0.120	0.000	0.000	0.113	0.111	0.240	0.052	0.023	0.002	0.000
	RSF [DZ23]	0.030	0.020	0.080	0.055	0.000	0.000	0.109	0.000	0.002	0.025	0.012	0.011	0.000
	SeMoLi [Sei+24] †	-	0.195	-	0.575	-	-	-	-	-	-	-	-	0.018
	LISO-CP	0.292	0.211	0.272	0.204	0.208	0.140	0.607	0.029	0.010	0.239	0.146	0.110	0.019
Self Train	Oyster-CP [Zha+23]	0.217	0.084	0.151	0.062	0.176	0.056	0.562	0.000	0.000	0.176	0.023	0.067	0.000
	Oyster-TF [Zha+23]	0.121	0.015	0.051	0.007	0.098	0.010	0.475	0.000	0.000	0.092	0.003	0.031	0.000
	LISO-CP	0.380	0.308	0.350	0.296	0.322	0.255	0.695	0.055	0.022	0.343	0.219	0.182	0.021
	LISO-TF	0.327	0.208	0.349	0.245	0.233	0.126	0.669	0.024	0.012	0.277	0.136	0.116	0.009

Table 6.2: Evaluation on WOD dataset: We evaluate using the protocol of [Sei+24; Naj+22], using an area of whole $100\text{ m} \times 40\text{ m}$ BEV grid around the ego vehicle, considering objects that move faster than 1 m s^{-1} to be moving (difficulty level L2). CP, TF: network architecture, in the first two lines trained supervised for comparison. †: Results taken from [Sei+24]. SF: LiDAR scene flow by SLIM, GF: ground truth LiDAR scene flow. For class-specific 3D detection scores see Appendix D.

sors and annotated with 3D object boxes. On AV2 and KITTI, we evaluate both 2D (in BEV space) and 3D box IoU at IoU thresholds of 0.3 and 0.5 in the area of $100 \times 100\text{ m}$ around the ego vehicle.

nuScenes: [Cae+20] is recorded using a Velodyne VLP32 LiDAR sensor. We evaluate using the nuScenes protocol.

6.3.2 Networks

We evaluate our proposed method with two state-of-the-art LiDAR object detection networks of different architecture: Centerpoint [YZK21] and Transfusion-L [Bai+22]. We do not modify the published network architectures or losses and train both networks with a batch size of four.

CenterPoint (CP): [YZK21] is based on 2D convolutions in BEV space. Object centers are represented as heatmap in the BEV, which are then reduced to a sensible amount of boxes using NMS.

Transfusion-L (TF): [Bai+22] is a Transformer based architecture, which is applied after initial encoding of the LiDAR point cloud into a 2D BEV feature tensor.

6.3.3 Baselines

Besides the obvious baseline of training the object detection networks supervised from scratch on ground truth from the dataset, we also compare to multiple strong unsupervised baselines. For details, see Section 3.3.

6 LiDAR-only Self-Supervised 3D Object Detection

	Movable (Moving & Still)				Car		Pedestrian		Cyclist	
	BEV-IOU		3D-IOU		BEV-IOU		BEV-IOU		BEV-IOU	
	AP@0.3↑	AP@0.5↑	AP@0.3↑	AP@0.5↑	AP@0.3↑	AP@0.5↑	AP@0.3↑	AP@0.5↑	AP@0.3↑	AP@0.5↑
CP [YZK21]	0.755	0.690	0.736	0.601	0.814	0.794	0.370	0.128	0.409	0.157
TF [Bai+22]	0.747	0.665	0.729	0.582	0.820	0.776	0.311	0.096	0.263	0.032
DBSCAN [Est+96]	0.023	0.002	0.010	0.000	0.026	0.005	0.000	0.000	0.064	0.007
RSF [DZ23]	0.029	0.019	0.029	0.011	0.066	0.049	0.000	0.000	0.192	0.043
Oyster-CP [Zha+23]	0.235	0.098	0.114	0.000	0.327	0.135	0.000	0.000	0.000	0.000
Oyster-TF [Zha+23]	0.273	0.088	0.128	0.000	0.364	0.121	0.000	0.000	0.019	0.000
LISO-CP	0.446	0.330	0.419	0.159	0.520	0.411	0.097	0.019	0.445	0.053
LISO-TF	0.361	0.207	0.294	0.036	0.425	0.297	0.084	0.014	0.348	0.003

Table 6.3: Evaluation on KITTI dataset: We evaluate on the forward facing field of view where GT annotations are available, but run inference on the whole 100×100 m BEV grid. Also note that flow annotations are not available for KITTI Object. CP, TF: network architecture, in the first two lines trained supervised for comparison.

RSF: We include RSF [DZ23] in our evaluation as strong representative of methods doing object distilling from motion cues. We ran experiments using the published code of the authors.

DBSCAN: This algorithm [Est+96] clusters points in the point cloud with similar 3D locations into objects. We additionally evaluate extending the cluster space to 6D by either using SLIM scene flow (SF) or ground-truth scene flow (GT). Thus, the "DBSCAN(SF)" baseline essentially corresponds to the raw detections used for our initial pseudo ground truth generation. We use the implementation from [Ped+11] with parameter values 1.0 for epsilon and 5 for the minimum number of points for a valid cluster.

SeMoLi: [Sei+24] is the most recent baseline. As it showed to outperform [Naj+22] as self-supervised object detection method we include only published results of [Sei+24] in our evaluation.

Oyster: We reimplemented [Zha+23] using PyTorch [Pas+19], and apply the proposed framework to the CenterPoint (Oyster-CP) and Transfusion-L (Oyster-TF) to be as close as possible to our method and evaluation. We verify our re-implementation, by reproducing the reported metrics by [Zha+23] on AV2, see Table 6.1.

6.3.4 Results

Quantitative Results: On all four datasets we see that our method consistently outperforms all self-supervised baselines in all metrics for *movable* objects, see Table 6.1, Table 6.2, and Table 6.3. Only SeMoLi [Sei+24] beats LISO on *moving* objects (see Table 6.2) but significantly drops below LISO’s performance on *movable* objects, hinting to difficulties with generalization. Our method does not suffer from this performance gap and performs nearly equally well on *movable* and *moving* objects. We also observe that, despite mostly better performance in the supervised case, Transfusion-L responds less favorable than Centerpoint to the self-supervised training approaches. We suspect that Centerpoint, due to its convolutional architecture and its lack of positional encoding

compared to Transfusion-L, is less susceptible to overfitting to where *moving* objects have been observed and where not to expect them.

We also observe that the nuScenes dataset seems to be the most challenging for the self-supervised methods, leading to the biggest gap between supervised and unsupervised object detection performance. We attribute this to the sparseness of the lidar sensor with only 32 layers. All self-supervised methods have difficulties estimating the orientation of objects (AOE on nuScenes, 2π periodic, see Table 6.1), which the IoU-based metrics cannot reveal.

Qualitative Results:

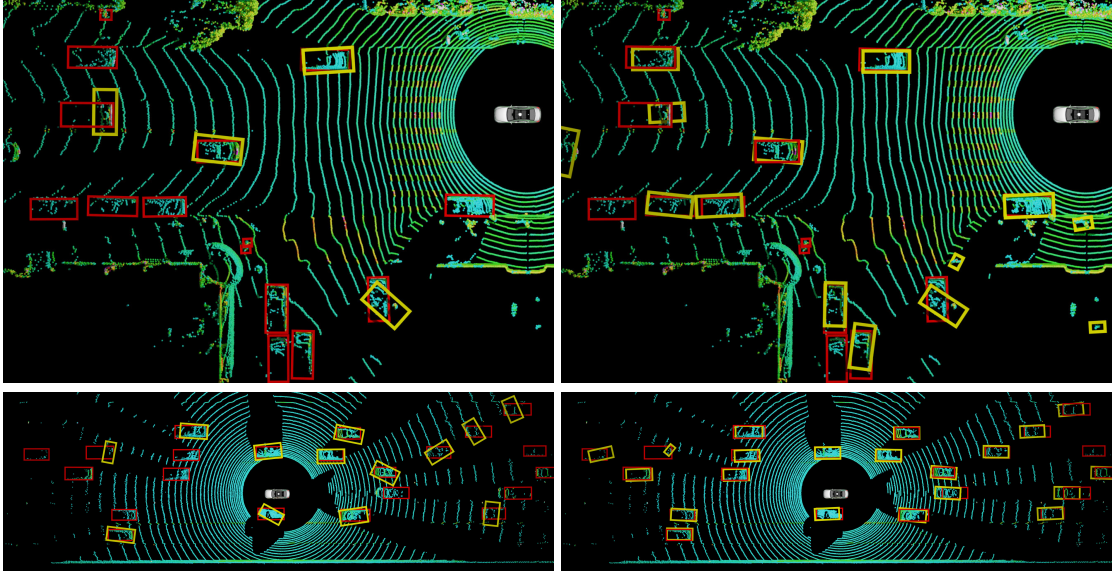


Figure 6.5: Qualitative Results on WOD. *Red* boxes are ground truth boxes, *yellow* boxes are predictions. **Left: Oyster-CP Right: LISO-CP**

In Fig. 6.5 we compare our method to the best-performing baseline, Oyster, using Centerpoint as network architecture. It can be seen, that our method has higher accuracy and predicts box orientations more accurately.

Fig. 6.6, in contrast, showcases an important failure-case and general problem of all unsupervised methods: ground truth, especially in the WOD dataset, is also annotated in very sparse and distant regions. It is very challenging to generate pseudo ground truth for such regions whereas supervised trainings do not have to generalize from *moving* to such *movable* (but very sparse) objects. This may be the main reason for a still noticeable gap in evaluation results between supervised and unsupervised methods.

6.3.5 Ablation Study

We investigated the influence of different choices for the components in our pipeline, see Table 6.4 and cf. Fig. 6.2.

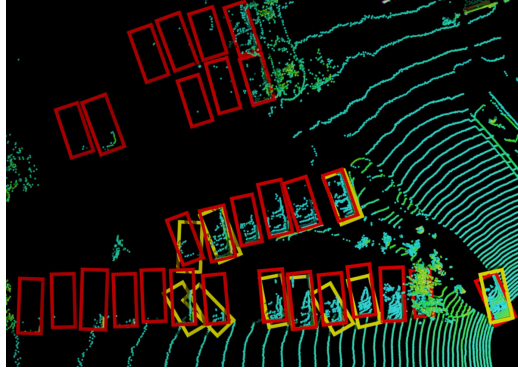


Figure 6.6: Missed ground truth boxes in the WOD dataset.

Motion Cues as Clustering Input: When adding motion cues to box clustering for Oyster we observe that detection performance already increases noticeably (comparing #3 to #4, Table 6.4). This demonstrates that self-supervised motion cues are a key ingredient to get high-quality initial pseudo ground truth as opposed to just using clustered 3D points.

Tag	Cluster Input	Method	Self Train	Track Optim.	AP@0.4(BEV)	AP@0.4(3D)
#1	P, SF	no tracker			0.177	0.086
#2	P, SF	no tracker	✓		0.201	0.131
#3	P	Oyster	✓		0.217	0.084
#4	P, SF	Oyster	✓		0.255	0.104
#5	P, SF	LISO(K, SF)			0.279	0.209
#6	P, SF	LISO(K, SF)	✓		0.360	0.290
#7	P, SF	LISO(K, SF)		✓	0.292	0.211
#8	P, SF	LISO(K, SF)	✓	✓	0.380	0.308
#9	P, SF	LISO(G, SF)	✓	✓	0.411	0.339
#10	P, GF	LISO(G, GF)	✓	✓	0.423	0.349

Table 6.4: Ablation study of different parts of our pipeline on WOD with Center-point [YZK21]. **Abbreviations:** P: points, SF: self-supervised *LiDAR* scene flow (SLIM), GF: ground truth *LiDAR* scene flow. K: KISS-ICP ego-motion, G: ground truth ego-motion.

Motion Cues for Tracking: Our scene-flow-based tracker is the biggest contributing factor to the success of our method: the ablation reveals that leveraging motion cues greatly improves tracking (comparing #4 to #6) and thus ultimately improves self-training, i.e. more accurate tracking allows for much more strictness when matching and filtering tracklets. The Oyster tracker uses a matching threshold of 5 m, our threshold is only 1.5 m. This strictness leads to higher quality pseudo ground truth.

Effect of Self Training: Even without using any tracker to filter tracklets self training has a benefit (comparing #1 to #2). This is due to the network generalizing to new instances and confidence thresholding the detections. However, having an additional way

to discard implausible detections (by tracking) amplifies the positive effect of self training (comparing #5 to #6 and #7 to #8), as it can better prevent false positives from entering the pseudo ground truth.

Track Optimization: The added benefit of track optimization is more independent of self training (comparing going from #5 to #7 with going from #6 to #8). Finally we notice, that with the combination of a strict tracker, track optimization and self training, the performance is very close to using ground truth flow and ego-motion (comparing #8 to #9 and #10).

Hyperparameters for Iterative Training: Fig. 6.4 demonstrates that the performance of our proposed method is relatively consistent across a range of hyperparameters. However, using too few steps between regeneration of pseudo ground truth ($s = 20k$) leads to degrading performance, because the network does not have enough time to generalize and stabilize using the pseudo ground truth, which then has negative effects during regeneration of the pseudo ground truth. The experiment reveals that doing multiple iterations of self-learning and dropping network weights to allow the network to adjust is beneficial for a good performance. We selected $r = 2$ and $s = 30k$ as a good compromise between performance and speed for all other experiments.

For more in-depth evaluations the reader is referred to the appendix in Appendix D.

6.4 Conclusion

We have proposed a novel framework for self-supervised 3D lidar object detection based on self-supervised **LiDAR** scene flow. Using simple clustering on **LiDAR** scene flow in combination with a novel flow-based tracker, we were able to generate pseudo ground truth with *moving* objects at high precision. This enabled us to bootstrap state-of-the-art 3D lidar object detectors, without using any human labels. The detector generalizes from detecting *moving* to detecting *movable* objects over multiple self-training iterations. Experiments revealed that our trajectory-regularized self-learning, based on our scene-flow-based tracker, is key to the success of our method. We have demonstrated the effectiveness of our approach on two state-of-the-art lidar object detectors as well as four **AD** datasets. All experiments were conducted with the same set of hyperparameters, demonstrating the robustness and wide applicability of our method. Our method achieves a significant improvement in the state-of-the-art of self-supervised lidar object detection on all four datasets. The biggest limitation of our approach is that our method is not able to distinguish different classes. Future work could be around generating pseudo ground truth for class labels, e.g. based on motion or size characteristics.

7 Conclusion

In this thesis, we investigated two methods capable of obtaining 3D bounding box annotations for LiDAR point clouds without the need for human labeling.

To this end, this thesis has presented a novel physically-based simulation framework for LiDAR point clouds for AD applications in Chapter 4. Our model simulates both the visibility of objects in the scene as well as the noise and artifacts present in real-world LiDAR data, based on physical parameters and properties of the sensor and the environment. For evaluation, we have compared our simulation model against strong baselines. Firstly, we have compared the simulated data against real-world data visually, showing that our simulation model is able to generate visually realistic LiDAR point clouds. Secondly, we have analyzed the noise characteristics of the simulated data using surfaces with known geometric properties, showing that our simulation model matches the noise characteristics of real-world data more closely than these baselines. Furthermore, we have used the physically-based simulation models and baselines to train and evaluate various DL models for LiDAR object detection, comparing their suitability as source of training data. Additionally, we have evaluated the simulators viability as a tool for validating real-world behavior of LiDAR object detectors in simulation. Unfortunately, we saw that despite higher visual and statistical similarity to real-world data, the physically-based simulation model did not lead to better results w.r.t. the baselines.

In Chapter 5, we presented a new method for self-supervised LiDAR scene flow estimation and motion segmentation. To this end, we introduced a novel network architecture, which allowed us to train and evaluate on larger point clouds than our baselines, eliminating the necessity of downsampling. Our method, using only raw LiDAR point cloud sequences and no human annotations, along with losses based on nearest neighbor matching and rigid cycle consistency, significantly outperformed the state-of-the-art for self-supervised LiDAR scene flow estimation at the time of publication. Additionally, we successfully generated a self-supervised training signal for motion segmentation solely from motion cues, which in turn improved the LiDAR scene flow estimation.

Finally, in Chapter 6 we introduced a novel self-supervised, trajectory-regularized self-training framework for 3D object detection in LiDAR point clouds, without the need for any human annotations. This is achieved by first estimating self-supervised LiDAR scene flow (cf. Chapter 5) and using it to bootstrap initial pseudo ground truth for moving objects by clustering the LiDAR scene flow, and performing box fitting, tracking, and track optimization. Subsequently, a LiDAR object detector is trained on this initial pseudo ground truth. Being a single frame based detector, it generalizes from *moving* to all *movable* objects in the scene, since it is unable to distinguish between *moving* and *stationary* objects. The pseudo ground truth is iteratively improved by tracking and optimizing the detected bounding boxes, and by retraining the detector. We demonstrated that our

method outperforms the state-of-the-art for self-supervised [LiDAR](#) object detection on four different datasets, using two different state-of-the-art object detectors and identical hyperparameters for all settings. This showcases the consistently good performance, wide applicability, robustness and potential cost savings of our method.

A Publications

- S. A. Baur, F. Moosmann, and A. Geiger. “LISO: Lidar-Only Self-supervised 3D Object Detection”. In: *Computer Vision - ECCV 2024 - 18th European Conference, Milan, Italy, September 29-October 4, 2024, Proceedings, Part LXXXVI*. Ed. by A. Leonardis, E. Ricci, S. Roth, O. Russakovsky, T. Sattler, and G. Varol. Vol. 15144. Lecture Notes in Computer Science. Springer, 2024, pp. 253–270. DOI: [10.1007/978-3-031-73016-0_15](https://doi.org/10.1007/978-3-031-73016-0_15). URL: https://doi.org/10.1007/978-3-031-73016-0_15
- S. A. Baur, D. J. Emmerichs, F. Moosmann, P. Pinggera, B. Ommer, and A. Geiger. “SLIM: Self-Supervised LiDAR Scene Flow and Motion Segmentation”. In: *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*. IEEE, 2021, pp. 13106–13116. DOI: [10.1109/ICCV48922.2021.01288](https://doi.org/10.1109/ICCV48922.2021.01288). URL: <https://doi.org/10.1109/ICCV48922.2021.01288>

Publications not discussed in this thesis:

- L. T. Triess, D. Peter, S. A. Baur, and J. M. Zöllner. “Quantifying Point Cloud Realism Through Adversarially Learned Latent Representations”. In: *Pattern Recognition - 43rd DAGM German Conference, DAGM GCPR 2021, Bonn, Germany, September 28 - October 1, 2021, Proceedings*. Ed. by C. Bauckhage, J. Gall, and A. G. Schwing. Vol. 13024. Lecture Notes in Computer Science. Springer, 2021, pp. 681–696. DOI: [10.1007/978-3-030-92659-5_44](https://doi.org/10.1007/978-3-030-92659-5_44). URL: https://doi.org/10.1007/978-3-030-92659-5_44
- F. Duffhauss and S. A. Baur. “PillarFlowNet: A Real-time Deep Multitask Network for LiDAR-based 3D Object Detection and Scene Flow Estimation”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2020, Las Vegas, NV, USA, October 24, 2020 - January 24, 2021*. IEEE, 2020, pp. 10734–10741. DOI: [10.1109/IROS45743.2020.9341002](https://doi.org/10.1109/IROS45743.2020.9341002). URL: <https://doi.org/10.1109/IROS45743.2020.9341002>
- S. A. Baur, F. Moosmann, S. Wirges, and C. B. Rist. “Real-time 3D LiDAR Flow for Autonomous Vehicles”. In: *2019 IEEE Intelligent Vehicles Symposium, IV 2019, Paris, France, June 9-12, 2019*. IEEE, 2019, pp. 1288–1295. DOI: [10.1109/IVS.2019.8814094](https://doi.org/10.1109/IVS.2019.8814094). URL: <https://doi.org/10.1109/IVS.2019.8814094>

B Appendix: Credits

This chapter describes the individual contributions of the authors to the projects SLIM [Bau+21] (Chapter 5) and LISO (Chapter 6) [BMG24].

B.1 SLIM: Self-Supervised LiDAR Scene Flow and Motion Segmentation

Stefan Andreas Baur and David Josef Emmerichs contributed equally to this work. Stefan Andreas Baur initiated the project, extended the RAFT backbone for scene flow and motion segmentation, and suggested using the Kabsch algorithm to improve the scene flow estimation on the static points. David Josef Emmerichs suggested the idea of using the discrepancy of the nearest neighbor loss between the raw flow and the rigid flow estimate as self-supervised training signal for motion segmentation, and finding the optimal classification threshold during training using exponential moving average. Frank Moosmann and Andreas Geiger provided feedback on the manuscript, and actively contributed to the manuscript, including proofreading. Peter Pinggera and Björn Ommer contributed by proofreading.

B.2 LISO: Lidar-only Self-Supervised 3D Object Detection

Stefan Andreas Baur initiated the project and suggested the idea of using self-supervised lidar scene flow for 3D object detection. Frank Moosmann and Andreas Geiger provided valuable feedback and comments on results and general research directions during the active phase of the project. Frank Moosmann and Andreas Geiger further contributed to the manuscript.

C Appendix: Self-Supervised LiDAR Scene Flow and Motion Segmentation

This appendix has been published in S. A. Baur, D. J. Emmerichs, F. Moosmann, P. Pinggera, B. Ommer, and A. Geiger. “SLIM: Self-Supervised LiDAR Scene Flow and Motion Segmentation”. In: *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*. IEEE, 2021, pp. 13106–13116. DOI: [10.1109/ICCV48922.2021.01288](https://doi.org/10.1109/ICCV48922.2021.01288). URL: <https://doi.org/10.1109/ICCV48922.2021.01288> [Bau+21].

C.1 Computation of Confidence Weights

Using the confidence logits we need to compute a set of weights for the above Kabsch algorithm. In our initial experiments, we first tried a softmax activation, however, we found it to be quite unstable and generalizes badly to unseen inputs. We attribute this behavior to the fact that a few outliers in the logit distribution contain almost all the weight after activation. Activating each logit independently with the sigmoid function alleviates the problem as all high logit values get mapped to roughly the same weight of 1. We now have to deal only with a numerical issue when all logits are quite low, e.g. below -100 for 32-bit floating-point-precision. Then the sigmoid function maps all values to exactly 0, leaving the Kabsch algorithm with nothing to optimize. We avoid this problem by normalizing the sum beforehand which theoretically does not change the result, however, we provide a fused computation for the sigmoid activation and normalization such that we have a numerically stable set of weights.

Defining $m_i = [\sigma(l_{cls,i}) \geq p_{stat}]$ as the mask for static points our goal is to compute

$$w_i = \frac{m_i \sigma(l_{wgt,i})}{\sum_{j|m_j=1} \sigma(l_{wgt,j})} \quad (\text{C.1})$$

with

$$\sigma(x) = \frac{1}{1 + e^{-x}} = e^x \frac{1}{1 + e^x} = e^x \sigma(-x) \quad . \quad (\text{C.2})$$

From the identity above it is easy to check that

$$\sigma(x) = e^{\min(x,0)} \sigma(|x|) = \begin{cases} e^0 \sigma(x) & \text{for } x \geq 0 \\ e^x \sigma(-x) & \text{for } x < 0 \end{cases} \quad (\text{C.3})$$

holds. Using this in (C.1) with $l_{wgt,i}^- = \min(l_{wgt,i}, 0)$ we have

$$w_i = \frac{m_i \sigma(|l_{wgt,i}|) \exp(l_{wgt,i}^-)}{\sum_{j|m_j=1} \sigma(|l_{wgt,j}|) \exp(l_{wgt,j}^-)} \quad . \quad (C.4)$$

In the above equation, the sigmoid activations are now numerically stable between 0.5 and 1. The well-known trick of shifting all exponents about $s = \max_{i|m_i=1} l_{wgt,i}^-$ in the above expression makes the whole expression numerically stable

$$w_i = \frac{m_i \sigma(|l_{wgt,i}|) \exp(l_{wgt,i}^- - s)}{\sum_{j|m_j=1} \sigma(|l_{wgt,j}|) \exp(l_{wgt,j}^- - s)} \quad (C.5)$$

as the largest term of the sum in the denominator is now at least as large as $\sigma(|s|)e^0 \geq 0.5$ and therefore the denominator is well above 0.

C.2 Moving Classification Threshold

The result of any classification task depends strongly on the chosen threshold based on the predicted scores. Fig. C.1 shows how differently trained networks generate different distributions for the classification scores and how some metrics vary when changing the threshold p_{thresh} for that score $\sigma(l_{cls,i})$. We are interested in optimizing the scene flow for the stationary and the moving points, therefore, we want to select the optimal threshold for the AEE 50-50 metric. As the classification is already used during training as a mask to improve the results of the resulting rigid transform $\mathbf{T}_r$ we need an online mechanism selecting this threshold. As the scores are bounded between 0 and 1 we do this by keeping a moving average of the AEE 50-50 metric m_k for $R = 100,000$ different thresholds p_k , resulting in different possible aggregated flows $\mathbf{f}_{agg,i}^{(k)}$.

For the supervised training, the update rule is

$$m'_k = m_k \alpha^{N_t} + (1 - \alpha^{N_t}) \frac{1}{\sum_i c_i} \sum_{i=1}^{N_t} c_i |\mathbf{f}_{gt,i} - \mathbf{f}_{agg,i}^{(k)}| \quad (C.6)$$

with N_t being the number of points in the currently processed point cloud, α being the decay factor of the exponential moving average per point, and c_i being a pointwise weighting factor which is 1 for stationary points and $\frac{N_{mov}}{N_{stat}}$ for moving points. The last ratio is measured across the whole training set and the weights, therefore, account for the fact that we are interested in an optimal AEE 50-50 compared to optimizing AEE. The update rule also accounts for the current size of the point cloud and gives more weight to large point clouds as it would happen if we would update for every point individually. As we initialize the moving metrics with 0, we compute an unbiased estimate through also tracking a bias counter $b'_k = b_k \alpha^{N_t} + (1 - \alpha^{N_t})$ and simply dividing the above computed metric by it: $\hat{m}_k = m_k / b_k$.

For the exponential moving average to "forget" after every 5,000 training iterations, we

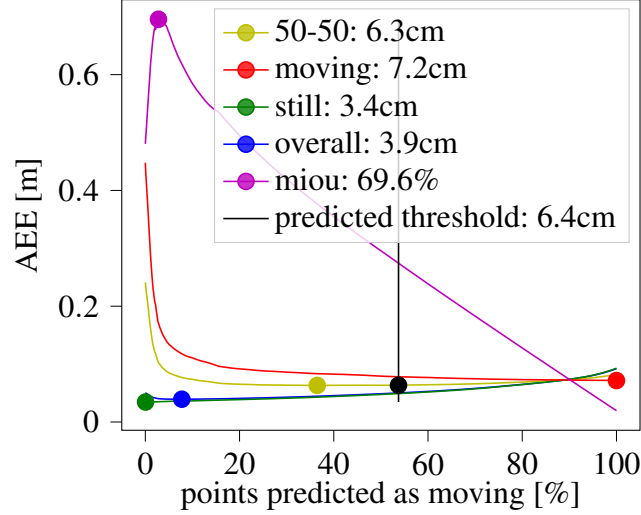


Figure C.1: Different metrics across changing thresholds for supervised training on nuScenes: The vertical line represents the moving threshold computed online during training. The yellow curve represents AEE 50-50 on the test set. Selecting all points as moving gives the best result for the moving AEE but the worst for stationary AEE. The opposite is true when classifying all points as stationary and using the improved rigid motion result. However, the moving AEE increases much more dramatically if points are classified wrongly as static and the optimal threshold is therefore around 40% moving points. Our moving average predicts just about over 50% moving, however, the AEE 50-50 is very flat in this area, and our achieved AEE 50-50 of 6.4 cm is just 1 mm lower than the optimal threshold.

set α to $1 - \frac{1}{5000\langle N_t \rangle}$, where $\langle N_t \rangle$ is the average point cloud size of the used dataset. Based on the unbiased estimates $\hat{m}_k$ we simply look for the best, therefore, smallest value and choose the corresponding threshold p_k as the optimal threshold $p_{thresh} = \operatorname{argmin}_{p_k} \hat{m}_k$. The whole update step can be easily vectorized and implemented efficiently using standard DL framework components including cumulative sums and sorting of the classification scores and is implemented similar to computing efficiently precision-recall-curves.

In the case of self-supervised trainings, we do not use the weights c_i as they already make use of labeled quantities, namely if a point is moving or stationary. We also can not use the ground truth flow and therefore replace the endpoint error term through the k-NN error term $e_i(\mathbf{f}_{aggr,i}^{(k)})$ defined in (5.4):

$$m'_k = m_k \alpha^{N_t} + (1 - \alpha^{N_t}) \sum_{i=1}^{N_t} |e_i(\mathbf{f}_{aggr,i}^{(k)})| \quad (\text{C.7})$$

Losing the controlled weighting and accurate error estimation negatively influences moving averages and the accuracy, however, we found it to work reasonably well. Fig. C.2 shows the same information as Fig. C.1 only for a self-supervised training corresponding to the third row of Table 5.7. Where in the supervised case the AEE 50-50 score from our

moving threshold computation is just 1 mm short compared to the optimal AEE 50-50, it is in the self-supervised case already more than 0.6 cm worse than the optimal threshold could have been.

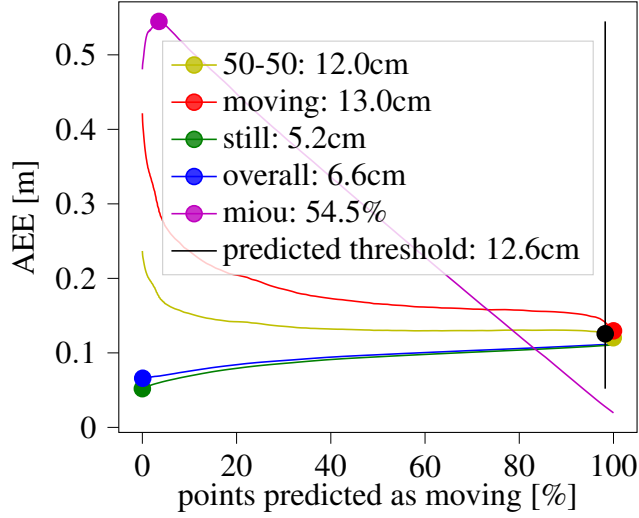


Figure C.2: Different metrics across changing thresholds for self-supervised training on nuScenes: The vertical line represents the moving threshold computed online during training using the k -NN residuals. The fact that stable self-supervised training criteria are more challenging to formulate for Scene Flow, leads to a larger gap between the optimal and learned threshold for self-supervised training.

C.3 Robust Outlier Metric

The outlier metric used by previous works [LQG19; Gu+19; Tis+20; Wu+20] defines a point as an outlier if its predicted flow vector has an endpoint error $>30\text{cm}$ or a relative error $>10\%$. However, we believe this metric has several drawbacks:

- The name and intuition about this metric are misleading. A predicted flow of 1.2cm length for a ground truth flow of only 1.0cm would be already considered an outlier where most people would classify this as normal noise.
- It also contradicts the accuracy metrics. The case mentioned above would not only be classified as an outlier but also as strict accurate ($<5\text{cm}$) and relaxed accurate ($<10\text{cm}$).
- The definition itself is not minimal. The case where the endpoint error is greater than 30cm but smaller than 10% basically does not exist, as this would require a ground truth flow larger than 3m, or accounting for 10Hz sample frequency $30\text{m/s}=108\text{km/h}$. This rarely occurs in the city-based datasets and therefore the 30cm condition is superfluous.

To avoid these drawbacks, we introduce a robust outlier metric ROutl which we report instead for the two newly investigated datasets, nuScenes and CARLA. It is defined as the relative portion of points that have a predicted endpoint error $>30\text{cm}$ and relative error $>30\%$. We believe it to not suffer from any of the drawbacks mentioned above:

- Requiring both the relative and absolute conditions to hold removes counting the small noise on an almost 0 ground truth flow to not be counted as an outlier. Also requiring a 30% error instead of 10% separates it well from the relaxed accuracy, following more closely the intuition of the name outlier.
- A flow classified as a robust outlier can now never be at the same time (relaxed) accurate as those definitions are mutually exclusive.
- The definition is now minimal. A prediction for a small flow ground truth flow of up to 1m with $30\% < 30\text{cm}$ needs to meet the absolute condition to be counted as an outlier, where a prediction of a large flow ground truth of above 1m or 36km/h needs to meet the relative condition.

Also, the robust outliers are counted as outliers by the old metric, so the robust outlier metric always reports a lower number. Table C.1 shows a comparison of the two metrics and how they behave in relation to each other. Especially for moving points, the Outl metric is very close to 100% for all methods and thus does not convey a lot of information. ROut discriminates better between the methods.

Table C.1: Self-supervised training & evaluation on nuScenes - comparison of metrics ROutl and Outl

	Moving		Static	
	ROutl↓	Outl↓	ROutl↓	Outl↓
Zero	0.5783	1.0000	0.6838	1.0000
PPWC [Wu+20]	0.3848	0.9395	0.1800	0.8508
PF [Tis+20]	0.9364	1.0000	0.0035	1.0000
Ours	0.1309	0.8125	0.0630	0.5286

C.4 Classification

We evaluate the classification using the following metrics (where TP: true positive, TN: true negative, FP: false positive, FN: false negative):

- accuracy: $\text{acc} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$
- sensitivity: $\text{sens} = \frac{\text{TP}}{\text{TP} + \text{FN}}$
- mean intersection over union: $\text{mIoU} = 0.5 \left(\frac{\text{TP}}{\text{TP} + \text{FP} + \text{FN}} + \frac{\text{TN}}{\text{TN} + \text{FP} + \text{FN}} \right)$

On KITTI-SF, a **mIoU** of 34% can be achieved by guessing that a point is moving with a probability of $p_{\text{mov,guess}} = 0.31$. This can be shown using the knowledge that roughly 40% of points in the dataset are moving ($p_{\text{gt}} = 0.4$), and from the optimality criteria for the binary classification: $\frac{\partial \text{mIoU}}{\partial p_{\text{mov,guess}}} = 0$ and $\frac{\partial (\text{IoU}_{\text{mov}} + \text{IoU}_{\text{still}})}{\partial p_{\text{mov,guess}}} = 0$. As already briefly mentioned in the paper, Table C.2 shows that our method achieves a **mIoU** score of 42.9% on KITTI-SF when trained self-supervised on KITTI-RL. The results show, that like for Scene Flow estimation, the domain gap from KITTI-RL to KITTI-SF seems to be the smallest and leads to the best performance achieved through self-supervision. The domain gap from simulated CARLA data to KITTI-SF is obviously larger than from nuScenes to KITTI-SF, which can explain the performance gap between the two datasets when evaluating on KITTI-SF for both the self-supervised and the supervised case. A few example images for motion segmentation by the network trained using self-supervision can be seen in Fig. C.3.

Table C.2: Classification results for self-supervised and supervised training for different training datasets, evaluated on KITTI-SF. Acc, **mIoU** & sens are the resulting values that are obtained when the threshold is optimized w.r.t. AEE 50-50 (in the self-supervised case via **k-NN** residuals as proxy). Opt acc & opt **mIoU** are scores that could be achieved if the threshold were selected w.r.t. optimal classification, regardless of Scene Flow estimation performance, a trade-off that is also discussed in Fig. C.1

	Train Dataset	acc	mIoU	sens	opt acc	opt mIoU
Self-	KITTI-RL	0.601	0.429	0.835	0.747	0.581
Super-	nuScenes	0.491	0.308	0.906	0.835	0.699
vised	CARLA	0.443	0.246	0.991	0.765	0.613
Super-	nuScenes	0.861	0.754	0.942	0.925	0.855
vised	CARLA	0.610	0.416	0.418	0.650	0.434

Once again, when comparing optimal scores and those optimized for AEE 50-50 in Table C.2, it becomes clear that there is a trade-off between optimal performance on the Scene Flow task (AEE 50-50) and optimal performance in terms of classification, as discussed in Section C.2.

C.5 Architecture & Parameter Details

C.5.1 Training

We use the RMSProp optimizer with 0.0001 for the learning rate, with warm-up and decay. For the first 2k steps of the training, the learning rate is exponentially increased, starting from 0.0 to 0.0001. After that, exponential decay is applied with a ratio of 0.5 every 60k steps. While we chose the batch size of 1 during training, we use a batch of point clouds to perform "forward" and "backward" predictions.

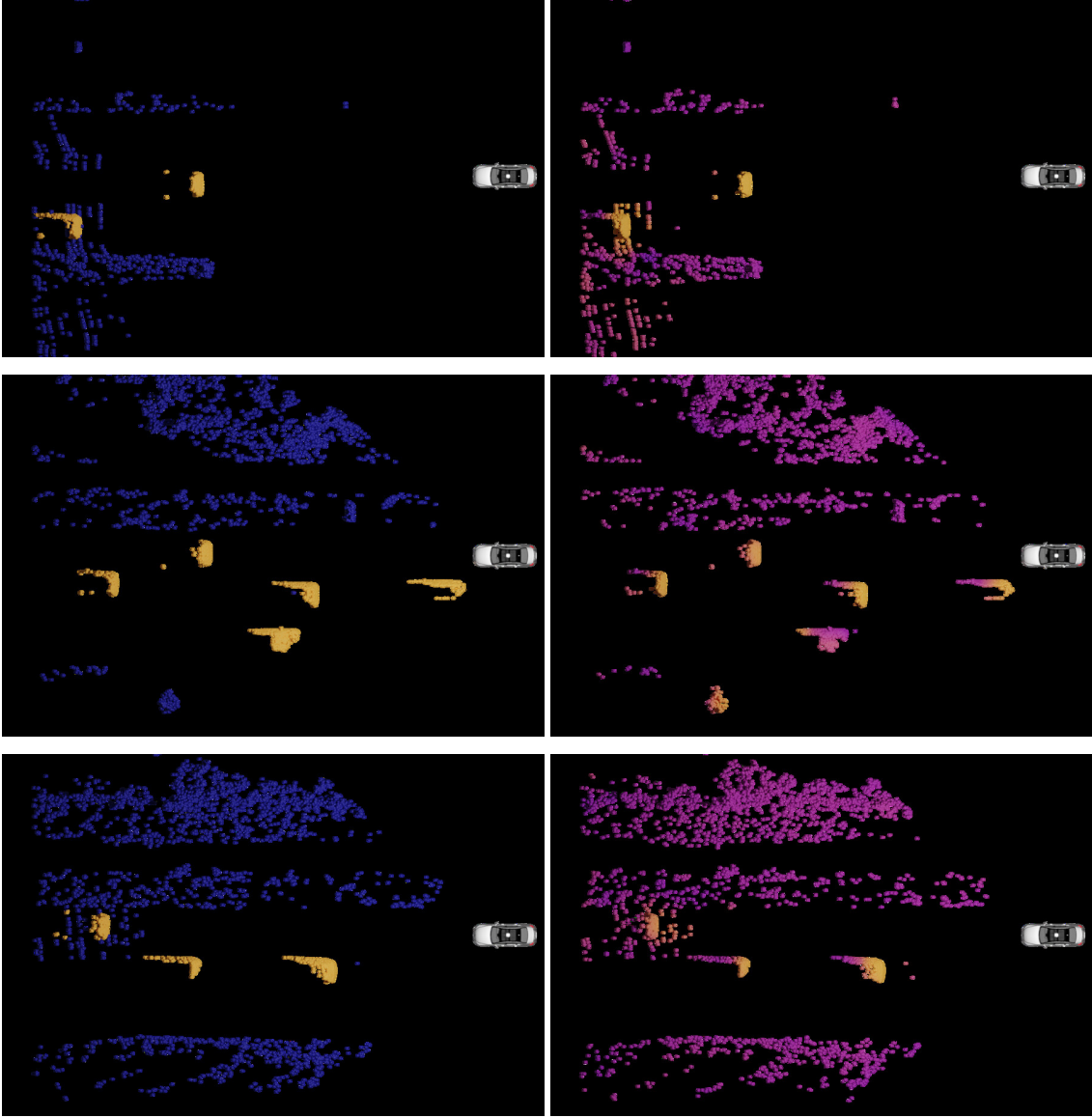


Figure C.3: Motion segmentation by our self-supervised network trained on KITTI-RL evaluated on KITTI-SF. Left: Ground truth motion segmentation, Right: Predicted motion segmentation by our network, brighter colors indicate higher dynamicity.

C.5.2 PillarFeatureNet

The PFN is used to create the BEV feature maps consumed $\mathcal{I}_t, \mathcal{I}_{t+1}$ by the RAFT backbone. We use the default configuration of the PFN introduced in [Lan+19] almost every aspect, with minor deviations described in the following: In their original paper, Lang et al. used batch normalization, but due to the small batch size in our experiment, we swap regular batch normalization for batch renormalization [Iof17]. This is then followed by ReLU activation, like in the original paper. In our experiments, the spatial BEV dimensions are $H = 640, W = 640$ and $C = 64$ feature channels are used: $\mathcal{I}_t, \mathcal{I}_{t+1} \in \mathbb{R}^{H \times W \times C}$, but any

reasonable combination of spatial and feature dimensions can work. We do not use the reflectivity/intensity channel of the point clouds, as the usage would break generalization when training on KITTI-RL and evaluating on KITTI-SF, where the network has to generalize from [LiDAR](#) point cloud data to stereo camera point cloud data.

C.5.3 RAFT

We use the parameters and configuration of the model called RAFT-S introduced in [\[TD20\]](#) whenever possible, for example, like in the original paper, the output of the context network (see [\[TD20\]](#)) has 160 channels and is divided into the initial hidden state h_0 with 96 channels and the encoded scene context with 64 channels. Like in the original RAFT-S as introduced by Teed et al. [\[TD20\]](#), we use 4 correlation levels with a correlation lookup radius of 3 each. Also, like in the original paper, the ConvGRU in the Update Block estimates flow at 1/8th of the original input resolution: $\mathcal{F} \in \mathbb{R}^{H/8 \times W/8 \times 2}$. As an extension to the original RAFT-S, we introduce logits $\mathcal{L}_{cls}$ and confidence weights $\mathcal{L}_{wgt}$, which are handled equivalently to the flow map, in the case of the logits with an additional encoder & decoder. The encoders in the Update Block use 2D convolutional layers with 64 and 32 channels each with ReLU activation. The Flow-Head (like in the RAFT-S paper version) and additional Classification-Head each use two convolutions, 128 filters each with ReLU activation. Unlike in the original paper, no mask for upsampling the predictions to full resolution is learned, instead bilinear upsampling is used. For training and inference, six iterations are used in the update block to refine the result.

C.6 Transferability of the SLIM Loss Framework onto PointPWC

Our ablation study shows an improvement of our RAFT network over the other baseline methods, without applying our novel loss framework (Table 7, row 1). In addition, it demonstrates that motion segmentation and Kabsch regularization for rigid static flow boost SLIM’s performance when trained self-supervised (row 2+3).

To investigate the relationship between the network architecture and the performance boost provided by our loss framework we present results of our SLIM loss applied to another model.

PoseFlowNet [\[Tis+20\]](#) already has a mechanism to deal with the rigid scene motion so our loss cannot be directly applied onto that architecture. However, PointPWC [\[Wu+20\]](#) has a suitable architecture to test the transferability of our proposed losses. We extend the PPWC architecture with the necessary logits to apply our loss framework (classification & confidence), as well as the necessary Kabsch regularization (from here on out referred to as PPWC-MOD), and apply the SLIM loss framework to each of the four pyramid levels. As the application of SLIM’s losses requires cyclic consistency as a self-supervised training signal, we additionally extend the training regime of PointPWC to do forward and backward scene flow estimation by adding an inference step for the time-reversed

input point clouds.

Table C.3 shows that the above-introduced modifications result in a worse performance than the baseline PPWC (row 1,4). Further investigations into this matter reveal that the cyclic training routine does not harmonize well with PPWC. Therefore, we present the results of two more training configurations: The PPWC trained on cyclic samples (without any other modification) (row 2) and the PPWC-MOD without cyclic sampling and training (row 3). These experiments show that SLIM produces slightly better results compared to the corresponding baselines for some metrics, but not for all. In the case of non-cyclic training, it is also shown, that there are only marginal differences. We conclude that SLIM’s performance boost also heavily depends on the chosen network architecture.

Table C.3: Ablation study on nuScenes for PPWC[Wu+20]. Flow directions during training: *f*: forward, *r*: reverse. *f+r*: training using both forward and reverse time. *f&r*: training using both forward and reverse time with explicit cyclic consistency between forward and reverse predictions.

*: SLIM loss without cyclic consistency

network arch.	loss framework	flow dirs training	Mov. AEE	Stat. AEE	50-50 AEE
PPWC [Wu+20]	orig	f	0.3539	1.1974	0.2756
PPWC [Wu+20]	orig	f+r	0.6432	0.5457	0.5945
PPWC-MOD	SLIM*	f	0.3837	0.1475	0.2656
PPWC-MOD	SLIM	f&r	0.4005	0.2284	0.3145

C.7 Runtimes

The runtimes of SLIM and the baselines are reported in Table C.4. Please note that SLIM does not make use of the optimizations reported in [TD20].

Table C.4: Runtimes as reported by the authors of each method. The runtime for RAFT-S has been measured using an implementation optimized for half-precision, SLIM (TensorFlow, 6 RAFT stages, unoptimized) takes 443 ms on a single Tesla V100 GPU.

Method	Runtime
PPWC [Wu+20]	117 ms
PF [Tis+20]	491 ms
RAFT-S [TD20]	50 ms
SLIM	443 ms

C.8 Failure Cases

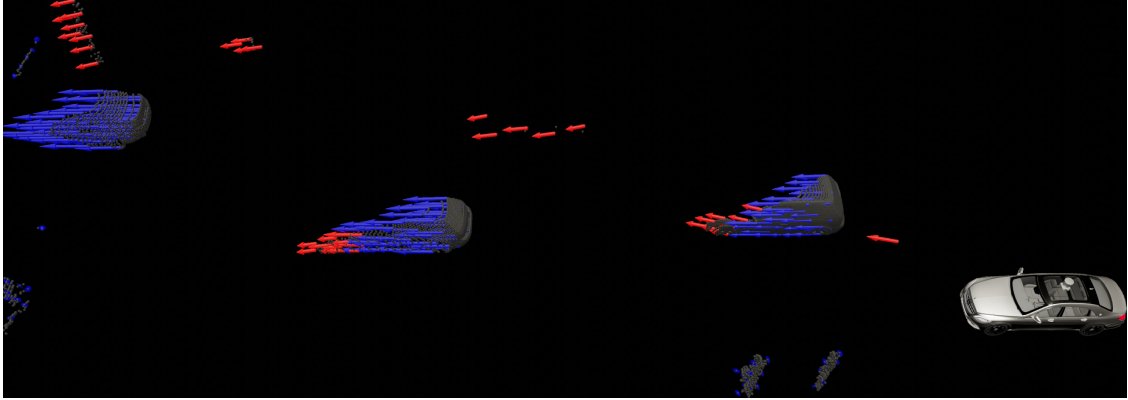


Figure C.4: Very dynamic scene leads to flow outliers on the static pole in the background (top left). (red: outlier, blue: accurate)

Our network predicts quite accurate static flow and thus has no problem distinguishing cars as moving even when having the same speed as the sensor as long as there are some static objects present. Most scenes in the NuScenes dataset have more than 90% of static points. Even for scenes with the lowest amount of static points of around 20% the flow estimates for the dominant dynamic objects is correct and only a few static elements fail, see Fig. C.4.

D Appendix: LiDAR-only Self-Supervised 3D Object Detection

This appendix has been published in S. A. Baur, F. Moosmann, and A. Geiger. “LISO: Lidar-Only Self-supervised 3D Object Detection”. In: *Computer Vision - ECCV 2024 - 18th European Conference, Milan, Italy, September 29-October 4, 2024, Proceedings, Part LXXXVI*. Ed. by A. Leonardis, E. Ricci, S. Roth, O. Russakovsky, T. Sattler, and G. Varol. Vol. 15144. Lecture Notes in Computer Science. Springer, 2024, pp. 253–270. DOI: [10.1007/978-3-031-73016-0_15](https://doi.org/10.1007/978-3-031-73016-0_15). URL: https://doi.org/10.1007/978-3-031-73016-0_5C_15 [BMG24].

D.1 Implementation Details

We run the networks Centerpoint [YZK21] and Transfusion-L [Bai+22] on 100×100 m BEV grids around the ego vehicle. We use NMS with a threshold of 0.1 (2D BEV IoU) for the detections. The optimizers, as well as their learning rate schedules are kept from the respective original implementations, but the schedules are shortened to match the lifecycle of the network weights during the iterative rounds of self-training. For the zero-shot generalization required by Oyster [Zha+23] after the first round, we found that starting from an initial BEV range of 50×50 m, and then extending to 100×100 m, gave the best results. For DBSCAN we used $\epsilon = 1.0$, and $\text{minPts} = 5$. We optimize all tracks using Adam optimizer with learning rate 0.1 for 2000 steps for a complete point cloud sequence batched (at the same time), which takes less than 2 s per nuScenes session on a Nvidia V100 GPU.

D.2 Self-supervised LiDAR Scene Flow

As mentioned in Section 6.3, we extend the BEV range of SLIM [Bau+21] from 70×70 m, 640×640 pixels to 120×120 m and 920×920 pixels, but make no further modifications to the network. This results in the state-of-the-art scene flow quality described in Table D.1.

The small performance gap of our method between using ground truth and SLIM LiDAR scene flow (comparing the last and the second-to-last row of Table 6.4) demonstrates that SLIM LiDAR scene flow has suitable quality for our method, and also that our method does not require absolutely perfect LiDAR scene flow estimates to work well.

Ground truth **LiDAR** scene flow is generated using the recorded vehicle egomotion for static points and the tracking information (bounding boxes) of moving objects.

D.3 Additional Ablations

Box-size Thresholds for Initial Pseudo Ground Truth: After fitting boxes to the initial **LiDAR** scene flow clusters, we discard abnormally-sized boxes, i.e. boxes that are smaller than a child or quite elongated cf. Section 6.2.2. Despite these thresholds being very permissive, having size limitations for the initial pseudo ground truth could potentially limit the applicability of our method. We therefore investigate how the performance of our method changes when omitting this constraint influences the performance in Table D.2. While the impact on the overall performance is very minor and looks promising, further investigations would be required to exclude the possibility that small children and long but thin vehicles might potentially be underrepresented in the dataset, which would lead to a similar effect on the metrics.

Omitting Weights Dropping During Self Training: We also investigate the influence of weights dropping on the overall performance during self-training: I.e. we keep periodic regeneration of pseudo ground truth, but the network weights are never dropped between the rounds of incremental self training. The negative impact on performance is more significant here. We believe dropping the weights helps the network to escape the overfitting to noise from the previous iteration of pseudo ground truth more easily than via weak negative gradients.

D.4 Performance of **LiDAR** Scene Flow Clustering on nuScenes

In the evaluation on nuScenes (see Table D.3), the worse performance of using DBSCAN [Est+96] clustering on ground truth **LiDAR** scene flow compared to using DBSCAN on SLIM

Train Data	Val Data	AEE(moving)[m] ↓	AEE(static)[m] ↓
AV2 Train	AV2 Val	0.079	0.075
KITTI Raw	KITTI Tracking	0.092	0.104
nuScenes Train	nuScenes Val	0.132	0.077
WOD Train	WOD Val	0.091	0.085

Table D.1: Lidar scene flow metrics of SLIM [Bau+21] on the datasets (evaluated on val split), for a **BEV** range of 120×120 m. Note that for KITTI, we only evaluate the forward-facing **FOV** which has been annotated with tracked objects. Objects faster than 1 ms^{-1} are considered moving. AEE refers to the average endpoint error across either all moving or static points.

Cluster Input	Method	Modification	AP@0.4 BEV	AP@0.4 3D
P, SF	LISO(K, SF)	-	0.380	0.308
P, SF	LISO(K, SF)	keep all cluster sizes	0.366	0.296
P, SF	LISO(K, SF)	never drop weights	0.334	0.261

Table D.2: Additional ablations for *LISO-CP* on WOD (Movable). We investigate the influence of omitting the dropping of weights during self training (“never drop weights”), and the influence of omitting the discarding of clusters based on the size constraints as described in Section 6.2.2 (“keep all cluster sizes”). *P*: point cloud, *SF*: self-supervised *LiDAR* scene flow (SLIM), *K*: KISS-ICP.

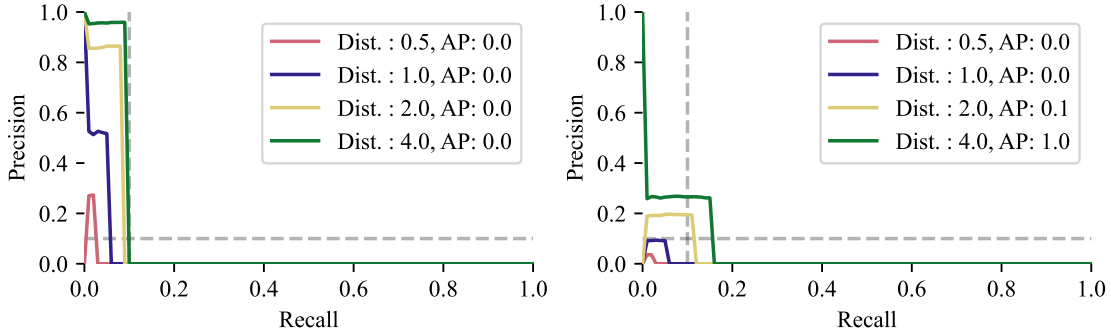


Figure D.1: Performance comparison of clustering ground truth *LiDAR* scene flow (left) and SLIM [Bau+21] *LiDAR* scene flow (right) on the nuScenes dataset. The methods are evaluated according to the official nuScenes protocol on the validation split. The dashed line represents the minimum threshold for precision and recall of 0.1, all results below these two thresholds are discarded. This leads to the surprising effect that the *AP* score is higher when using SLIM *LiDAR* scene flow, but this is only a result of the clipping dictated by the nuScenes evaluation protocol.

LiDAR scene flow is surprising. However, this peculiar effect is explained by Fig. D.1, which shows the full precision-recall curves, generated using the official nuScenes protocol on the validation split [Cae+20]. The nuScenes protocol uses minimum precision and recall value thresholds of 0.1, discarding all results below these thresholds. As mentioned in Section 6.2.2, we assign confidence score of 1.0 to all clusters discovered by DBSCAN. This causes all detections generated using DBSCAN on ground truth *LiDAR* scene flow to be discarded.

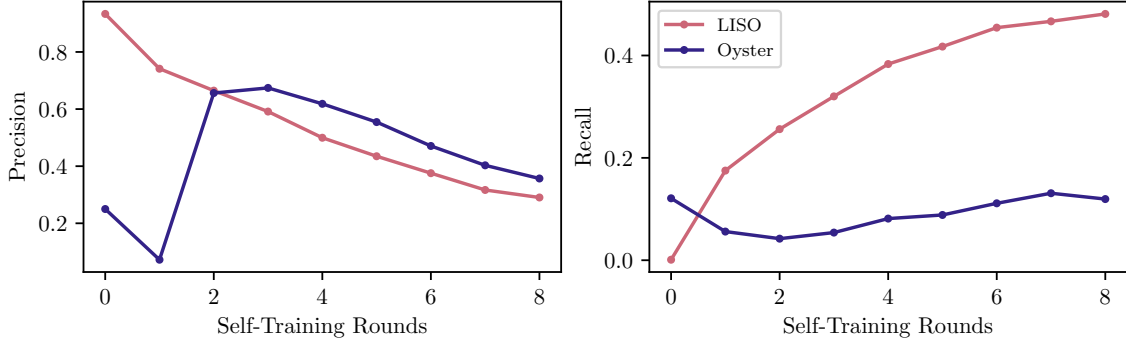


Figure D.2: Precision and recall of the (tracked) pseudo ground truth generated by Oyster and LISO over the course of self-training of Centerpoint on WOD (training split). Precision and recall are computed like in the **AP** metrics used in Fig. 6.4 and Table 6.2, i.e. true positives are occurrences where the **BEV IoU** between ground truth and predicted boxes is greater than 0.4, but at a specific confidence threshold: For Oyster, we use the reported value from the publication $c = 0.4$ [Zha+23]. For LISO, we use $c = 0.3$ and only discard the learned weights every other round, as stated in Section 6.2.2. Note that the dip in Oyster’s performance at round 1 stems from the zero-shot generalization, where the network is tasked to generalize from the training on the initial pseudo ground truth generated on the smaller **BEV** range to the full, previously unseen **BEV** range, going from $50 \times 50\text{m}$ to $100 \times 100\text{m}$.

D.5 Quality of Pseudo Ground Truth during Iterative Self-Training

One critical aspect of iterative self-training is the quality of pseudo ground truth on the training dynamics, as depicted in Fig. D.2. Finding the right balance between precision and recall in the pseudo ground truth is crucial for achieving optimal performance during self-training iterations: In our experiments, we find that having initially a small subset of high precision training samples is superior to having a larger set with higher recall but worse precision, because it allows the model to learn from a smaller but more reliable set of labeled data. A larger set of pseudo ground truth that is collected with less rigorous clustering, tracking and filtering, includes more noisy and mislabeled data. As discussed in [Zha+23; You+22], the limited model capacity does prevent the model from overfitting to the inconsistent noises in the pseudo ground truth to some extent and the model generalizes mostly to the objects of interest, but in our experiments, higher quality pseudo ground truth with less noise ultimately leads to better performance. Motion cues (i.e. egomotion and **LiDAR** scene flow) are the superior clustering and tracking input signal, allowing our method to generate much cleaner initial pseudo ground truth when compared to Oyster, which we also demonstrate in our ablation in Table 6.4. Fig. D.3 additionally visually demonstrates the difference between using **LiDAR** scene flow for initial pseudo ground truth creation and just using point clouds (Oyster) on an example point cloud: As expected, using **LiDAR** scene flow leads to fewer false positives in the initial pseudo ground truth.

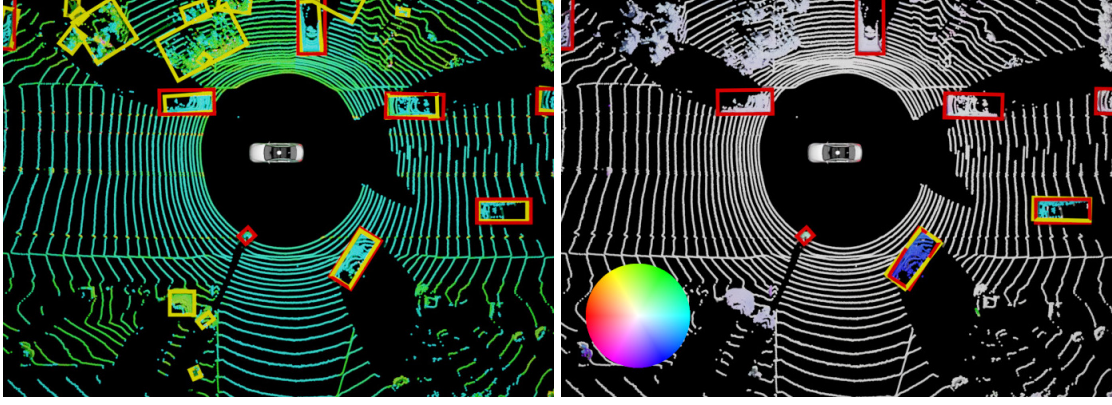


Figure D.3: Clustering results for the initial pseudo ground truth generation on WOD. *Red* boxes are ground truth boxes, *yellow* are predictions. **Left:** *Oyster* Clustering result on points, with high recall but low precision. **Right:** *LISO* Clustering result on points and *SLIM LiDAR* scene flow, resulting in high precision pseudo ground truth (*LISO*). Points are colored according to flow direction and magnitude.

D.6 Qualitative Results

For more qualitative comparisons besides Fig. D.4 or Fig. 6.5, we kindly refer the reader to the video accompanying this supplement.

D.7 Quantitative Results

In Table D.4 and Table D.3 we have more detailed metrics for WOD and nuScenes. Please note that the models get a high penalty on the Nuscenes Detection Score (NDS), because they cannot distinguish object classes and therefore score an Average Attribute Error of 1.0.

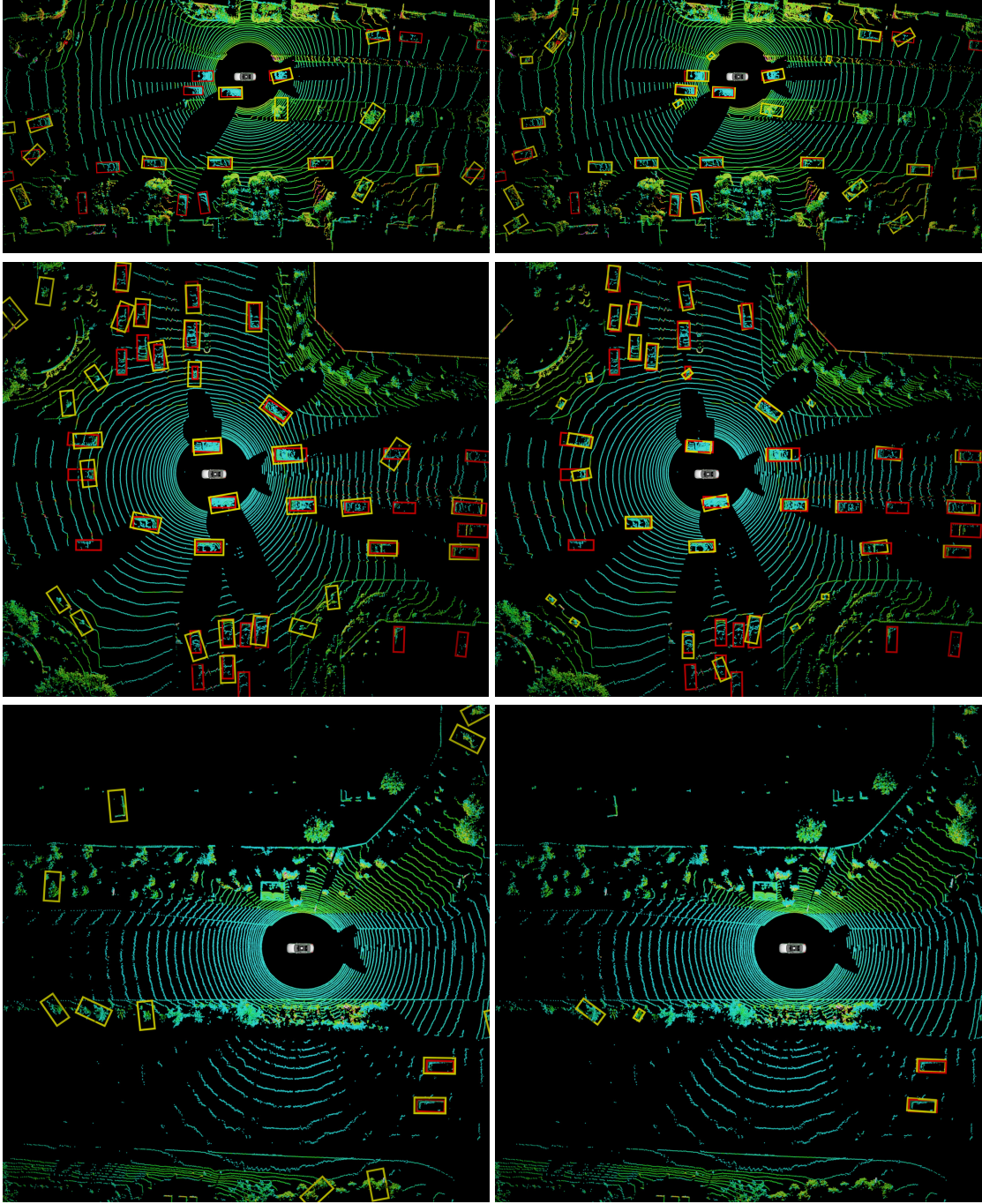


Figure D.4: Qualitative Results on WOD. *Red* boxes are ground truth boxes, *yellow* are predictions. **Left: OYSTER-CP Right: LISO-CP** Both methods struggle to some extent with false positive detections, but Oyster much more so, despite using the higher confidence threshold. We attribute this to the fact that Oyster has noisier initial pseudo ground truth, which leads to wrong training signals.

	method	AP $\uparrow$	NDS $\uparrow$	ATE $\downarrow$	AOE $\downarrow$	ASE $\downarrow$
GT	CP [YZK21]	0.484	0.524	0.357	0.560	0.263
	TF [Bai+22]	0.627	0.614	0.287	0.501	0.207
Unsup.	DBSCAN [Est+96]	0.008	0.109	0.987	0.171	0.962
	DBSCAN(SF)	0.003	0.106	1.186	0.082	0.952
	DBSCAN(GF)	0.000	0.000	1.000	1.0	1.0
	RSF [DZ23]	0.019	0.183	0.774	1.003	0.507
Self Train	Oyster-CP [Zha+23]	0.091	0.215	0.784	1.514	0.521
	Oyster-TF [Zha+23]	0.093	0.233	0.708	1.564	0.448
	LISO-CP	0.109	0.224	0.750	1.062	0.409
	LISO-TF	0.134	0.270	0.628	0.938	0.408

Table D.3: Full evaluation results on nuScenes dataset: We compare **LISO** with two different network architectures (TF [Bai+22], CP [YZK21]) against different baselines and also give supervised training results as reference (two top rows). Along the AP score we report the nuScenes detection score NDS, which is a combination of the AP score, average translation, orientation, scale, attribute error/score (ATE, AOE, ASE, AEE respectively). All models get a high penalty on the Nuscenes Detection Score (NDS), because they cannot distinguish object classes and therefore score an Average Attribute Error of 1.0. Note that nuScenes uses a minimum precision and recall threshold of 0.1, and since the recall of GT flow clustering is lower than 0.1, all results are clipped away. SF: LiDAR scene flow by SLIM, GF: ground truth LiDAR scene flow.

		Movable		Moving		Still		Vehicle		Pedestrian		Cyclist	
		AP@0.4		AP@0.4		AP@0.4		AP@0.4		AP@0.4		AP@0.4	
		BEV	3D	BEV	3D	BEV	3D	BEV	3D	BEV	3D	BEV	3D
GT	CP [YZK21]	0.765	0.684	0.721	0.624	0.735	0.657	0.912	0.841	0.513	0.413	0.134	0.094
	TF [Bai+22]	0.746	0.723	0.714	0.668	0.733	0.710	0.918	0.899	0.457	0.429	0.216	0.187
Unsupervised	DBSCAN [Est+96]	0.027	0.008	0.009	0.000	0.027	0.006	0.184	0.048	0.002	0.000	0.001	0.000
	DBSCAN(SF)	0.026	0.010	0.064	0.041	0.000	0.000	0.073	0.046	0.010	0.006	0.009	0.006
	DBSCAN(GF)	0.114	0.071	0.318	0.120	0.000	0.000	0.113	0.075	0.111	0.063	0.240	0.151
	RSF [DZ23]	0.030	0.020	0.080	0.055	0.000	0.000	0.109	0.074	0.000	0.000	0.002	0.000
	SeMoLi [Sei+24] †	-	0.195	-	0.575	-	-	-	-	-	-	-	-
	LISO-CP	0.292	0.211	0.272	0.204	0.208	0.140	0.607	0.440	0.029	0.009	0.010	0.004
Self Train	Oyster-CP [Zha+23]	0.217	0.084	0.151	0.062	0.176	0.056	0.562	0.204	0.000	0.000	0.000	0.000
	Oyster-TF [Zha+23]	0.121	0.015	0.051	0.007	0.098	0.010	0.475	0.058	0.000	0.000	0.000	0.000
	LISO-CP	0.380	0.308	0.350	0.296	0.322	0.255	0.695	0.543	0.055	0.037	0.022	0.016
	LISO-TF	0.327	0.208	0.349	0.245	0.233	0.126	0.669	0.408	0.024	0.008	0.012	0.005

Table D.4: Full evaluation results on WOD dataset: We evaluate using the protocol of [Sei+24; Naj+22], using an area of whole $100\text{ m} \times 40\text{ m}$ BEV grid around the ego vehicle, considering objects that move faster than 1 m/s to be moving (difficulty level L2). CP, TF: network architecture, in the first two lines trained supervised for comparison. †: Results taken from [Sei+24]. SF: LiDAR scene flow by SLIM, GF: ground truth LiDAR scene flow.

Bibliography

- [AG24] M.-B. AG. *Mercedes-Benz DRIVE PILOT*. 2024. URL: <https://www.mercedes-benz.de/passengercars/technology/drive-pilot.html#sensor-and-lidar> (visited on 05/26/2024).
- [Alv+03] A. Alvandpour, R. Krishnamurthy, D. Eckerbert, S. Apperson, B. Bloechel, and S. Borkar. “A 3.5GHz 32mW 150nm multiphase clock generator for high-performance microprocessors”. In: *2003 IEEE International Solid-State Circuits Conference, 2003. Digest of Technical Papers. ISSCC. 2003*, 112–482 vol.1. DOI: [10.1109/ISSCC.2003.1234230](https://doi.org/10.1109/ISSCC.2003.1234230).
- [Ama+01] M.-C. Amann, T. Bosch, M. Lescure, R. Myllyla, and M. Rioux. “Laser ranging: a critical review of usual techniques for distance measurement”. In: *OPTICAL ENGINEERING -BELLINGHAM- INTERNATIONAL SOCIETY FOR OPTICAL ENGINEERING-* 40 (2001), pp. 10–19. ISSN: 0091-3286. URL: <https://www.tib.eu/de/suchen/id/BLSE%3ARN091016540>.
- [Ami+22] M. Amini, V. Feofanov, L. Pauletto, E. Devijver, and Y. Maximov. “Self-Training: A Survey”. In: *CoRR* abs/2202.12040 (2022). arXiv: [2202.12040](https://arxiv.org/abs/2202.12040). URL: <https://arxiv.org/abs/2202.12040>.
- [Ami+23] M.-R. Amini, V. Feofanov, L. Pauletto, L. Hadjadj, E. Devijver, and Y. Maximov. *Self-Training: A Survey*. 2023. arXiv: [2202.12040](https://arxiv.org/abs/2202.12040) [cs.LG].
- [Bai+22] X. Bai, Z. Hu, X. Zhu, Q. Huang, Y. Chen, H. Fu, and C. Tai. “TransFusion: Robust LiDAR-Camera Fusion for 3D Object Detection with Transformers”. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*. IEEE, 2022, pp. 1080–1089. DOI: [10.1109/CVPR52688.2022.00116](https://doi.org/10.1109/CVPR52688.2022.00116). URL: <https://doi.org/10.1109/CVPR52688.2022.00116>.
- [Bal+23] R. Balestrieri, M. Ibrahim, V. Sobal, A. Morcos, S. Shekhar, T. Goldstein, F. Bordes, A. Bardes, G. Mialon, Y. Tian, A. Schwarzschild, A. G. Wilson, J. Geiping, Q. Garrido, P. Fernandez, A. Bar, H. Pirsiavash, Y. LeCun, and M. Goldblum. “A Cookbook of Self-Supervised Learning”. In: *CoRR* abs/2304.12210 (2023). DOI: [10.48550/ARXIV.2304.12210](https://doi.org/10.48550/ARXIV.2304.12210). arXiv: [2304.12210](https://arxiv.org/abs/2304.12210). URL: <https://doi.org/10.48550/arXiv.2304.12210>.
- [Bau+19] S. A. Baur, F. Moosmann, S. Wirges, and C. B. Rist. “Real-time 3D LiDAR Flow for Autonomous Vehicles”. In: *2019 IEEE Intelligent Vehicles Symposium, IV 2019, Paris, France, June 9-12, 2019*. IEEE, 2019, pp. 1288–1295. DOI: [10.1109/IVS.2019.8814094](https://doi.org/10.1109/IVS.2019.8814094). URL: <https://doi.org/10.1109/IVS.2019.8814094>.

- [Bau+21] S. A. Baur, D. J. Emmerichs, F. Moosmann, P. Pinggera, B. Ommer, and A. Geiger. “SLIM: Self-Supervised LiDAR Scene Flow and Motion Segmentation”. In: *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*. IEEE, 2021, pp. 13106–13116. DOI: [10.1109/ICCV48922.2021.01288](https://doi.org/10.1109/ICCV48922.2021.01288). URL: <https://doi.org/10.1109/ICCV48922.2021.01288>.
- [Beh+19a] A. Behl, D. Paschalidou, S. Donné, and A. Geiger. “PointFlowNet: Learning Representations for Rigid Motion Estimation From Point Clouds”. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 2019, pp. 7962–7971. DOI: [10.1109/CVPR.2019.00815](https://openaccess.thecvf.com/content%5C_CVPR%5C_2019/html/Behl%5C_PointFlowNet%5C_Learning%5C_Representations%5C_for%5C_Rigid%5C_Motion%5C_Estimation%5C_From%5C_Point%5C_Clouds%5C_CVPR%5C_2019%5C_paper.html). URL: http://openaccess.thecvf.com/content%5C_CVPR%5C_2019/html/Behl%5C_PointFlowNet%5C_Learning%5C_Representations%5C_for%5C_Rigid%5C_Motion%5C_Estimation%5C_From%5C_Point%5C_Clouds%5C_CVPR%5C_2019%5C_paper.html.
- [Beh+19b] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall. “SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences”. In: *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*. IEEE, 2019, pp. 9296–9306. DOI: [10.1109/ICCV.2019.00939](https://doi.org/10.1109/ICCV.2019.00939). URL: <https://doi.org/10.1109/ICCV.2019.00939>.
- [BH16] S. Bechtold and B. Höfle. “HELIOS: A MULTI-PURPOSE LIDAR SIMULATION FRAMEWORK FOR RESEARCH, PLANNING AND TRAINING OF LASER SCANNING OPERATIONS WITH AIRBORNE, GROUND-BASED MOBILE AND STATIONARY PLATFORMS”. In: *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences III-3* (June 2016), pp. 161–168. DOI: [10.5194/isprsannals-III-3-161-2016](https://doi.org/10.5194/isprsannals-III-3-161-2016).
- [BM18] D. Bolkas and A. Martinez. In: *Journal of Applied Geodesy* 12.1 (2018), pp. 109–127. DOI: [doi:10.1515/jag-2017-0034](https://doi.org/10.1515/jag-2017-0034). URL: <https://doi.org/10.1515/jag-2017-0034>.
- [BMG24] S. A. Baur, F. Moosmann, and A. Geiger. “LISO: Lidar-Only Self-supervised 3D Object Detection”. In: *Computer Vision - ECCV 2024 - 18th European Conference, Milan, Italy, September 29-October 4, 2024, Proceedings, Part LXXXVI*. Ed. by A. Leonardis, E. Ricci, S. Roth, O. Russakovsky, T. Sattler, and G. Varol. Vol. 15144. Lecture Notes in Computer Science. Springer, 2024, pp. 253–270. DOI: [10.1007/978-3-031-73016-0_15](https://doi.org/10.1007/978-3-031-73016-0_15). URL: https://doi.org/10.1007/978-3-031-73016-0_15.
- [BMK13] T. Basha, Y. Moses, and N. Kiryati. “Multi-view Scene Flow Estimation: A View Centered Variational Approach”. In: *Int. J. Comput. Vis.* 101.1 (2013), pp. 6–21. DOI: [10.1007/s11263-012-0542-7](https://doi.org/10.1007/s11263-012-0542-7). URL: <https://doi.org/10.1007/s11263-012-0542-7>.

- [BML18] P. Bideau, R. R. Menon, and E. G. Learned-Miller. “MoA-Net: Self-supervised Motion Segmentation”. In: *Computer Vision - ECCV 2018 Workshops - Munich, Germany, September 8-14, 2018, Proceedings, Part VI*. Ed. by L. Leal-Taixé and S. Roth. Vol. 11134. Lecture Notes in Computer Science. Springer, 2018, pp. 715–730. DOI: [10.1007/978-3-030-11024-6_55](https://doi.org/10.1007/978-3-030-11024-6_55). URL: https://doi.org/10.1007/978-3-030-11024-6_55.
- [Bro98] R. Brooks. “Artificial Life and Real Robots”. In: (June 1998).
- [Cae+20] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom. “nuScenes: A Multimodal Dataset for Autonomous Driving”. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*. IEEE, 2020, pp. 11618–11628. DOI: [10.1109/CVPR42600.2020.01164](https://doi.org/10.1109/CVPR42600.2020.01164). URL: <https://doi.org/10.1109/CVPR42600.2020.01164>.
- [Car+20] M. Caron, I. Misra, J. Mairal, P. Goyal, P. Bojanowski, and A. Joulin. “Unsupervised Learning of Visual Features by Contrasting Cluster Assignments”. In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin. 2020. URL: <https://proceedings.neurips.cc/paper/2020/hash/70feb62b69f16e0238f741fab228fec2-Abstract.html>.
- [CC17] M. Chevrier and G. Campanella. “How to build a LIDAR system with a time-to-digital converter”. In: *Texas Instruments - Analog Applications Journal* (Mar. 2017).
- [Che+23] Z. Chen, Y. Luo, Z. Wang, M. Baktashmotlagh, and Z. Huang. “Revisiting Domain-Adaptive 3D Object Detection by Reliable, Diverse and Class-balanced Pseudo-Labeling”. In: *IEEE/CVF International Conference on Computer Vision, ICCV 2023, Paris, France, October 1-6, 2023*. IEEE, 2023, pp. 3691–3703. DOI: [10.1109/ICCV51070.2023.00344](https://doi.org/10.1109/ICCV51070.2023.00344). URL: <https://doi.org/10.1109/ICCV51070.2023.00344>.
- [Che+24] Y. Chen, M. Mancini, X. Zhu, and Z. Akata. “Semi-Supervised and Unsupervised Deep Visual Learning: A Survey”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 46.3 (2024), pp. 1327–1347. DOI: [10.1109/TPAMI.2022.3201576](https://doi.org/10.1109/TPAMI.2022.3201576). URL: <https://doi.org/10.1109/TPAMI.2022.3201576>.
- [CK02] R. L. Carceroni and K. N. Kutulakos. “Multi-View Scene Capture by Surf-fel Sampling: From Video Streams to Non-Rigid 3D Motion, Shape and Reflectance”. In: *Int. J. Comput. Vis.* 49.2-3 (2002), pp. 175–214. DOI: [10.1023/A:1020145606604](https://doi.org/10.1023/A:1020145606604). URL: <https://doi.org/10.1023/A:1020145606604>.

- [Com21] E. Commission. *Vehicle safety – type approval of cars and vans with emergency lane keeping systems (ELKS)*. 2021. URL: https://ec.europa.eu/info/law/better-regulation/have-your-say/initiatives/12503-Vehicle-safety-type-approval-of-cars-and-vans-with-emergency-lane-keeping-systems-ELKS-_en (visited on 05/26/2024).
- [CSS19] Y. Chen, C. Schmid, and C. Sminchisescu. “Self-Supervised Learning With Geometric Constraints in Monocular Video: Connecting Flow, Depth, and Camera”. In: *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*. IEEE, 2019, pp. 7062–7071. DOI: [10.1109/ICCV.2019.00716](https://doi.org/10.1109/ICCV.2019.00716). URL: <https://doi.org/10.1109/ICCV.2019.00716>.
- [CT82] R. L. Cook and K. E. Torrance. “A reflectance model for computer graphics”. In: *ACM Transactions on Graphics (TOG)* 1.1 (1982), pp. 7–24.
- [CYS15] Z. Cheng, Q. Yang, and B. Sheng. “Deep Colorization”. In: *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*. IEEE Computer Society, 2015, pp. 415–423. DOI: [10.1109/ICCV.2015.55](https://doi.org/10.1109/ICCV.2015.55). URL: <https://doi.org/10.1109/ICCV.2015.55>.
- [DB20] F. Duffhauss and S. A. Baur. “PillarFlowNet: A Real-time Deep Multitask Network for LiDAR-based 3D Object Detection and Scene Flow Estimation”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2020, Las Vegas, NV, USA, October 24, 2020 - January 24, 2021*. IEEE, 2020, pp. 10734–10741. DOI: [10.1109/IROS45743.2020.9341002](https://doi.org/10.1109/IROS45743.2020.9341002). URL: <https://doi.org/10.1109/IROS45743.2020.9341002>.
- [Dev+19] J. Devlin, M. Chang, K. Lee, and K. Toutanova. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*. Ed. by J. Burstein, C. Doran, and T. Solorio. Association for Computational Linguistics, 2019, pp. 4171–4186. DOI: [10.18653/v1/N19-1423](https://doi.org/10.18653/v1/N19-1423). URL: <https://doi.org/10.18653/v1/n19-1423>.
- [Dew+16] A. Dewan, T. Caselitz, G. D. Tipaldi, and W. Burgard. “Motion-based detection and tracking in 3D LiDAR scans”. In: *2016 IEEE International Conference on Robotics and Automation, ICRA 2016, Stockholm, Sweden, May 16-21, 2016*. Ed. by D. Kragic, A. Bicchi, and A. D. Luca. IEEE, 2016, pp. 4508–4513. DOI: [10.1109/ICRA.2016.7487649](https://doi.org/10.1109/ICRA.2016.7487649). URL: <https://doi.org/10.1109/ICRA.2016.7487649>.

- [Dos+15a] A. Dosovitskiy, P. Fischer, E. Ilg, P. Häusser, C. Hazırbaş, V. Golkov, P. v.d. Smagt, D. Cremers, and T. Brox. “FlowNet: Learning Optical Flow with Convolutional Networks”. In: *IEEE International Conference on Computer Vision (ICCV)*. 2015. URL: <http://lmb.informatik.uni-freiburg.de/Publications/2015/DFIB15>.
- [Dos+15b] A. Dosovitskiy, P. Fischer, E. Ilg, P. Häusser, C. Hazırbaş, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox. “FlowNet: Learning Optical Flow with Convolutional Networks”. In: *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*. IEEE Computer Society, 2015, pp. 2758–2766. DOI: [10.1109/ICCV.2015.316](https://doi.org/10.1109/ICCV.2015.316). URL: <https://doi.org/10.1109/ICCV.2015.316>.
- [Dos+17a] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. “CARLA: An Open Urban Driving Simulator”. In: *Proceedings of the 1st Annual Conference on Robot Learning*. 2017, pp. 1–16.
- [Dos+17b] A. Dosovitskiy, G. Ros, F. Codevilla, A. M. López, and V. Koltun. “CARLA: An Open Urban Driving Simulator”. In: *1st Annual Conference on Robot Learning, CoRL 2017, Mountain View, California, USA, November 13-15, 2017, Proceedings*. Vol. 78. Proceedings of Machine Learning Research. PMLR, 2017, pp. 1–16. URL: <http://proceedings.mlr.press/v78/dosovitskiy17a.html>.
- [DZ23] D. Deng and A. Zakhor. “RSF: Optimizing Rigid Scene Flow From 3D Point Clouds Without Labels”. In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. Jan. 2023, pp. 1277–1286.
- [ELW20] J. Eichenholz, L. Lingvay, and D. Welford. “LiDAR System Operating at 1200-1400nm”. United States Patent 20200076152. Mar. 2020. URL: <https://www.freepatentsonline.com/20200076152.pdf>.
- [Epi19] Epic Games. *Unreal Engine*. Version 4.22.1. Apr. 25, 2019. URL: <https://www.unrealengine.com>.
- [Esk24] G. Eskandar. “An Empirical Study of the Generalization Ability of Lidar 3D Object Detectors to Unseen Domains”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2024, pp. 23815–23825.
- [Est+96] M. Ester, H. Kriegel, J. Sander, and X. Xu. “A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise”. In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), Portland, Oregon, USA*. Ed. by E. Simoudis, J. Han, and U. M. Fayyad. AAAI Press, 1996, pp. 226–231. URL: <http://www.aaai.org/Library/KDD/1996/kdd96-037.php>.

- [FRS01] K. P. F.R.S. “LIII. On lines and planes of closest fit to systems of points in space”. In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (1901), pp. 559–572. DOI: [10.1080/14786440109462720](https://doi.org/10.1080/14786440109462720).
- [FV10] M. Fellendorf and P. Vortisch. “Microscopic Traffic Flow Simulator VIS-SIM”. In: *Fundamentals of Traffic Simulation*. Ed. by J. Barceló. New York, NY: Springer New York, 2010, pp. 63–93. ISBN: 978-1-4419-6142-6. DOI: [10.1007/978-1-4419-6142-6_2](https://doi.org/10.1007/978-1-4419-6142-6_2). URL: https://doi.org/10.1007/978-1-4419-6142-6_2.
- [Gai+16] A. Gaidon, Q. Wang, Y. Cabon, and E. Vig. “Virtual Worlds as Proxy for Multi-Object Tracking Analysis”. In: *CoRR* abs/1605.06457 (2016). arXiv: [1605.06457](https://arxiv.org/abs/1605.06457). URL: <http://arxiv.org/abs/1605.06457>.
- [Gei+13] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. “Vision meets robotics: The KITTI dataset”. In: *Int. J. Robotics Res.* 32.11 (2013), pp. 1231–1237. DOI: [10.1177/0278364913491297](https://doi.org/10.1177/0278364913491297). URL: <https://doi.org/10.1177/0278364913491297>.
- [GL10] C. Glennie and D. D. Lichti. “Static calibration and analysis of the velodyne HDL-64E S2 for high accuracy mobile scanning”. In: *Remote Sensing* 2.6 (2010), pp. 1610–1624. ISSN: 20724292. DOI: [10.3390/rs2061610](https://doi.org/10.3390/rs2061610).
- [GLU12a] A. Geiger, P. Lenz, and R. Urtasun. “Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite”. In: *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*. 2012, pp. 3354–3361.
- [GLU12b] A. Geiger, P. Lenz, and R. Urtasun. “Are we ready for autonomous driving? The KITTI vision benchmark suite”. In: *2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012*. IEEE Computer Society, 2012, pp. 3354–3361. DOI: [10.1109/CVPR.2012.6248074](https://doi.org/10.1109/CVPR.2012.6248074). URL: <https://doi.org/10.1109/CVPR.2012.6248074>.
- [Gmb21] I. A. S. GmbH. *ibeo LUX 4L / ibeo LUX 8L / ibeo LUX HD Datasheet*. Ibeo Automotive Systems GmbH. 2021.
- [God+19] C. Godard, O. M. Aodha, M. Firman, and G. J. Brostow. “Digging Into Self-Supervised Monocular Depth Estimation”. In: *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*. IEEE, 2019, pp. 3827–3837. DOI: [10.1109/ICCV.2019.00393](https://doi.org/10.1109/ICCV.2019.00393). URL: <https://doi.org/10.1109/ICCV.2019.00393>.
- [Goj+21] Z. Gojcic, O. Litany, A. Wieser, L. J. Guibas, and T. Birdal. “Weakly Supervised Learning of Rigid 3D Scene Flow”. In: *CoRR* abs/2102.08945 (2021). arXiv: [2102.08945](https://arxiv.org/abs/2102.08945). URL: <https://arxiv.org/abs/2102.08945>.

- [Goo+14] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio. “Generative Adversarial Nets”. In: *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger. 2014, pp. 2672–2680. URL: <https://proceedings.neurips.cc/paper/2014/hash/5ca3e9b122f61f8f06494c97b1afccf3-Abstract.html>.
- [Gsc+11] M. Gschwandtner, R. Kwitt, A. Uhl, and W. Pree. “BlenSor: Blender Sensor Simulation Toolbox”. In: *Advances in Visual Computing*. Ed. by G. Bebis, R. Boyle, B. Parvin, D. Koracin, S. Wang, K. Kyungnam, B. Benes, K. Moreland, C. Borst, S. DiVerdi, C. Yi-Jen, and J. Ming. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 199–208. ISBN: 978-3-642-24031-7.
- [Gsc12] M. Gschwandtner. *BlenSor - Blender Sensor Simulation Toolkit*. <https://github.com/mgschwan/blensor>. 2012.
- [Gsc13] M. Gschwandtner. “Support Framework for Obstacle Detection on Autonomous Trains”. PhD thesis. Department of Computer Sciences, University of Salzburg, 2013.
- [Gu+19] X. Gu, Y. Wang, C. Wu, Y. J. Lee, and P. Wang. “HPLFlowNet: Hierarchical Permutohedral Lattice FlowNet for Scene Flow Estimation on Large-Scale Point Clouds”. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 2019, pp. 3254–3263. DOI: [10.1109/CVPR.2019.00337](https://doi.org/10.1109/CVPR.2019.00337). URL: http://openaccess.thecvf.com/content%5C_CVPR%5C_2019/html/Gu%5C_HPLFlowNet%5C_Hierarchical%5C_Permutohedral%5C_Lattice%5C_FlowNet%5C_for%5C_Scene%5C_Flow%5C_Estimation%5C_on%5C_CVPR%5C_2019%5C_paper.html.
- [Gui+22] B. Guillard, S. Vemprala, J. K. Gupta, O. Miksik, V. Vineet, P. Fua, and A. Kapoor. “Learning to Simulate Realistic LiDARs”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2022, Kyoto, Japan, October 23-27, 2022*. IEEE, 2022, pp. 8173–8180. DOI: [10.1109/IROS47612.2022.9981120](https://doi.org/10.1109/IROS47612.2022.9981120). URL: <https://doi.org/10.1109/IROS47612.2022.9981120>.
- [Har+21] A. W. Harley, Y. Zuo, J. Wen, A. Mangal, S. Potdar, R. Chaudhry, and K. Fragkiadaki. “Track, Check, Repeat: An EM Approach to Unsupervised Tracking”. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*. Computer Vision Foundation / IEEE, 2021, pp. 16581–16591. DOI: [10.1109/CVPR46437.2021.01631](https://doi.org/10.1109/CVPR46437.2021.01631). URL: https://openaccess.thecvf.com/content/CVPR2021/html/Harley%5C_Track%5C_Check%5C_Repeat%5C_An%5C_EM%5C_Approach%5C_to%5C_Unsupervised%5C_Tracking%5C_CVPR%5C_2021%5C_paper.html.

- [HCW19] B. Hurl, K. Czarnecki, and S. L. Waslander. “Precise Synthetic Image and LiDAR (PreSIL) Dataset for Autonomous Vehicle Perception”. In: *CoRR* abs/1905.00160 (2019). arXiv: 1905.00160. URL: <http://arxiv.org/abs/1905.00160>.
- [HD07] F. Huguet and F. Devernay. “A Variational Method for Scene Flow Estimation from Stereo Sequences”. In: *IEEE 11th International Conference on Computer Vision, ICCV 2007, Rio de Janeiro, Brazil, October 14-20, 2007*. IEEE Computer Society, 2007, pp. 1–7. DOI: 10.1109/ICCV.2007.4409000. URL: <https://doi.org/10.1109/ICCV.2007.4409000>.
- [Hen+19] D. Hendrycks, M. Mazeika, S. Kadavath, and D. Song. “Using Self-Supervised Learning Can Improve Model Robustness and Uncertainty”. In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. Ed. by H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. B. Fox, and R. Garnett. 2019, pp. 15637–15648. URL: <https://proceedings.neurips.cc/paper/2019/hash/a2b15837edac15df90721968986f7f8e-Abstract.html>.
- [Her+01] A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. Salesin. “Image analogies”. In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 2001, Los Angeles, California, USA, August 12-17, 2001*. Ed. by L. Pocock. ACM, 2001, pp. 327–340. DOI: 10.1145/383259.383295. URL: <https://doi.org/10.1145/383259.383295>.
- [HH19] K. Hassani and M. Haley. “Unsupervised Multi-Task Feature Learning on Point Clouds”. In: *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*. IEEE, 2019, pp. 8159–8170. DOI: 10.1109/ICCV.2019.00825. URL: <https://doi.org/10.1109/ICCV.2019.00825>.
- [HJA20] J. Ho, A. Jain, and P. Abbeel. “Denoising Diffusion Probabilistic Models”. In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin. 2020. URL: <https://proceedings.neurips.cc/paper/2020/hash/4c5bcfec8584af0d967f1ab10179ca4b-Abstract.html>.
- [Hof+18] J. Hoffman, E. Tzeng, T. Park, J. Zhu, P. Isola, K. Saenko, A. A. Efros, and T. Darrell. “CyCADA: Cycle-Consistent Adversarial Domain Adaptation”. In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. 2018, pp. 1994–2003. URL: <http://proceedings.mlr.press/v80/hoffman18a.html>.

- [HR20] J. Hur and S. Roth. “Self-Supervised Monocular Scene Flow Estimation”. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*. IEEE, 2020, pp. 7394–7403. DOI: [10.1109/CVPR42600.2020.00742](https://doi.org/10.1109/CVPR42600.2020.00742). URL: <https://doi.org/10.1109/CVPR42600.2020.00742>.
- [HS06] G. E. Hinton and R. R. Salakhutdinov. “Reducing the Dimensionality of Data with Neural Networks”. In: *Science* 313.5786 (2006), pp. 504–507. DOI: [10.1126/science.1127647](https://doi.org/10.1126/science.1127647). eprint: <https://www.science.org/doi/pdf/10.1126/science.1127647>. URL: <https://www.science.org/doi/abs/10.1126/science.1127647>.
- [Inc15] T. I. Incorporated. “TDC7200 Time-to-Digital Converter for Time-of-Flight Applications in LIDAR, Magnetostrictive and Flow Meters”. In: (Feb. 2015). Accessed: 2024-08-03. URL: <https://www.ti.com/document-viewer/tdc7200/datasheet>.
- [Ins18] T. Instruments. “LIDAR Pulsed Time of Flight Reference Design”. In: January (2018), pp. 1–25. URL: <https://www.ti.com/lit/ug/tiducm1b/tiducm1b.pdf>.
- [Iof17] S. Ioffe. “Batch Renormalization: Towards Reducing Minibatch Dependence in Batch-Normalized Models”. In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. Ed. by I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett. 2017, pp. 1945–1953. URL: <https://proceedings.neurips.cc/paper/2017/hash/c54e7837e0cd0ced286cb5995327d1ab-Abstract.html>.
- [Jai+15] M. Jaimez, M. Souiai, J. G. Jiménez, and D. Cremers. “A primal-dual framework for real-time dense RGB-D scene flow”. In: *IEEE International Conference on Robotics and Automation, ICRA 2015, Seattle, WA, USA, 26-30 May, 2015*. IEEE, 2015, pp. 98–104. DOI: [10.1109/ICRA.2015.7138986](https://doi.org/10.1109/ICRA.2015.7138986). URL: <https://doi.org/10.1109/ICRA.2015.7138986>.
- [Jan+18] J. Janai, F. Güney, A. Ranjan, M. J. Black, and A. Geiger. “Unsupervised Learning of Multi-Frame Optical Flow with Occlusions”. In: *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XVI*. Ed. by V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss. Vol. 11220. Lecture Notes in Computer Science. Springer, 2018, pp. 713–731. DOI: [10.1007/978-3-030-01270-0_42](https://doi.org/10.1007/978-3-030-01270-0_42). URL: https://doi.org/10.1007/978-3-030-01270-0_42.
- [Kab76] W. Kabsch. “A solution for the best rotation to relate two sets of vectors”. en. In: *Theoretical and General Crystallography* 32.5 (1976), p. 4.

- [Kar13] B. Karis. *Real Shading in Unreal Engine 4*. Tech. rep. Epic Games, 2013. URL: http://blog.selfshadow.com/publications/s2013-shading-course/karis/s2013_pbs_epic_notes_v2.pdf (visited on 02/05/2016).
- [Kau+21] P. Kaur, S. Taghavi, Z. Tian, and W. Shi. *A Survey on Simulators for Testing Self-Driving Cars*. 2021. arXiv: [2101.05337](https://arxiv.org/abs/2101.05337) [cs.R0].
- [KER20] Y. Kittenplon, Y. C. Eldar, and D. Raviv. “FlowStep3D: Model Unrolling for Self-Supervised Scene Flow Estimation”. In: *CoRR* abs/2011.10147 (2020). arXiv: [2011.10147](https://arxiv.org/abs/2011.10147). URL: <https://arxiv.org/abs/2011.10147>.
- [KH04] N. Koenig and A. Howard. “Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. Sendai, Japan, Sept. 2004, pp. 2149–2154.
- [Kim19] M. Kim. “Airborne Waveform Lidar Simulator Using the Radiative Transfer of a Laser Pulse”. In: *Applied Sciences* 9.12 (2019), p. 2452.
- [KK11] S. Kurtti and J. Kostamovaara. “An Integrated Laser Radar Receiver Channel Utilizing a Time-Domain Walk Error Compensation Scheme”. In: *IEEE Transactions on Instrumentation and Measurement* 60.1 (Jan. 2011), pp. 146–157. DOI: [10.1109/TIM.2010.2047663](https://doi.org/10.1109/TIM.2010.2047663).
- [KNK17] S. Kurtti, J. Nissinen, and J. Kostamovaara. “A Wide Dynamic Range CMOS Laser Radar Receiver With a Time-Domain Walk Error Compensation Scheme”. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 64.3 (Mar. 2017), pp. 550–561. DOI: [10.1109/TCSI.2016.2619762](https://doi.org/10.1109/TCSI.2016.2619762).
- [KST04] E. Knott, J. Schaeffer, and M. Tulley. *Radar Cross Section*. Radar, Sonar and Navigation Series. Institution of Engineering and Technology, 2004. ISBN: 9781891121258. URL: <https://books.google.de/books?id=j7hdXhgwsw4C>.
- [Kur12] S. Kurti. *Integrated receiver channel and timing discrimination circuits for a pulsed time-of-flight laser rangefinder*. 2012, p. 45. ISBN: 9789526200453.
- [Lab19] L. E. A. R. C. A. P. Lab. *LGSVL Simulator*. <https://github.com/lgsvl/simulator.git>. 2019.
- [Lan+19] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom. “PointPillars: Fast Encoders for Object Detection From Point Clouds”. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 2019, pp. 12697–12705. DOI: [10.1109/CVPR.2019.01298](https://doi.org/10.1109/CVPR.2019.01298). URL: http://openaccess.thecvf.com/content%5C_CVPR%5C_2019/html/Lang%5C_PointPillars%5C_Fast%5C_Encoders%5C_for%5C_Object%5C_Detection%5C_From%5C_Point%5C_Clouds%5C_CVPR%5C_2019%5C_paper.html.

- [Lee+19] K. Lee, G. Ros, J. Li, and A. Gaidon. “SPIGAN: Privileged Adversarial Learning from Simulation”. In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. 2019. URL: <https://openreview.net/forum?id=rkxoNnC5FQ>.
- [Li08] M. Li. *Jitter, Noise, and Signal Integrity at High-Speed*. Prentice Hall Modern Semiconductor Design Series. Prentice Hall, 2008. ISBN: 9780132429610. URL: <https://books.google.de/books?id=JRAfAQAAIAAJ>.
- [Lia+21] Z. Liang, Z. Zhang, M. Zhang, X. Zhao, and S. Pu. “RangeIoUDet: Range Image Based Real-Time 3D Object Detector Optimized by Intersection Over Union”. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*. Computer Vision Foundation / IEEE, 2021, pp. 7140–7149. DOI: [10.1109/CVPR46437.2021.00706](https://doi.org/10.1109/CVPR46437.2021.00706). URL: https://openaccess.thecvf.com/content/CVPR2021/html/Liang%5C_RangeIoUDet%5C_Range%5C_Image%5C_Based%5C_Real-Time%5C_3D%5C_Object%5C_Detector%5C_Optimized%5C_by%5C_CVPR%5C_2021%5C_paper.html.
- [Liu+19] P. Liu, M. R. Lyu, I. King, and J. Xu. “SelFlow: Self-Supervised Learning of Optical Flow”. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 2019, pp. 4571–4580. DOI: [10.1109/CVPR.2019.00470](https://doi.org/10.1109/CVPR.2019.00470). URL: http://openaccess.thecvf.com/content%5C_CVPR%5C_2019/html/Liu%5C_SelFlow%5C_Self-Supervised%5C_Learning%5C_of%5C_Optical%5C_Flow%5C_CVPR%5C_2019%5C_paper.html.
- [LKC17] W. Lotter, G. Kreiman, and D. D. Cox. “Deep Predictive Coding Networks for Video Prediction and Unsupervised Learning”. In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL: <https://openreview.net/forum?id=B1ewdt9xe>.
- [Lop+18] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wießner. “Microscopic Traffic Simulation using SUMO”. In: *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018. URL: <https://elib.dlr.de/124092/>.
- [LQG19] X. Liu, C. R. Qi, and L. J. Guibas. “FlowNet3D: Learning Scene Flow in 3D Point Clouds”. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 2019, pp. 529–537. DOI: [10.1109/CVPR.2019.00062](https://doi.org/10.1109/CVPR.2019.00062). URL: http://openaccess.thecvf.com/content%5C_CVPR%5C_2019/html/Liu%5C_FlowNet3D%5C_Learning%5C_Scene%5C_Flow%5C_in%5C_3D%5C_Point%5C_Clouds%5C_CVPR%5C_2019%5C_paper.html.

- [LTC19] H. Lai, Y. Tsai, and W. Chiu. “Bridging Stereo Matching and Optical Flow via Spatiotemporal Correspondence”. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 2019, pp. 1890–1899. DOI: [10.1109/CVPR.2019.00199](https://doi.org/10.1109/CVPR.2019.00199). URL: http://openaccess.thecvf.com/content%5C_CVPR%5C_2019/html/Lai%5C_Bridging%5C_Stereo%5C_Matching%5C_and%5C_Optical%5C_Flow%5C_via%5C_Spatiotemporal%5C_Correspondence%5C_CVPR%5C_2019%5C_paper.html.
- [Luo+20] C. Luo, Z. Yang, P. Wang, Y. Wang, W. Xu, R. Nevatia, and A. L. Yuille. “Every Pixel Counts ++: Joint Learning of Geometry and Motion with 3D Holistic Understanding”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 42.10 (2020), pp. 2624–2641. DOI: [10.1109/TPAMI.2019.2930258](https://doi.org/10.1109/TPAMI.2019.2930258). URL: <https://doi.org/10.1109/TPAMI.2019.2930258>.
- [LYB19] X. Liu, M. Yan, and J. Bohg. “MeteorNet: Deep Learning on Dynamic 3D Point Cloud Sequences”. In: *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*. IEEE, 2019, pp. 9245–9254. DOI: [10.1109/ICCV.2019.00934](https://doi.org/10.1109/ICCV.2019.00934). URL: <https://doi.org/10.1109/ICCV.2019.00934>.
- [Man+20] S. Manivasagam, S. Wang, K. Wong, W. Zeng, M. Sazanovich, S. Tan, B. Yang, W.-C. Ma, and R. Urtasun. “LiDARsim: Realistic LiDAR Simulation by Leveraging the Real World”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 11167–11176.
- [May+16] N. Mayer, E. Ilg, P. Häusser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox. “A Large Dataset to Train Convolutional Networks for Disparity, Optical Flow, and Scene Flow Estimation”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pp. 4040–4048. DOI: [10.1109/CVPR.2016.438](https://doi.org/10.1109/CVPR.2016.438). URL: <https://doi.org/10.1109/CVPR.2016.438>.
- [Mey+19] G. P. Meyer, A. Laddha, E. Kee, C. Vallespi-Gonzalez, and C. K. Wellington. “LaserNet: An Efficient Probabilistic 3D Object Detector for Autonomous Driving”. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 2019, pp. 12677–12686. DOI: [10.1109/CVPR.2019.01296](https://doi.org/10.1109/CVPR.2019.01296). URL: http://openaccess.thecvf.com/content%5C_CVPR%5C_2019/html/Meyer%5C_LaserNet%5C_An%5C_Efficient%5C_Probabilistic%5C_3D%5C_Object%5C_Detector%5C_for%5C_Autonomous%5C_Driving%5C_CVPR%5C_2019%5C_paper.html.

- [MF10] F. Moosmann and T. Fraichard. “Motion Estimation from Range Images in Dynamic Outdoor Scenes”. In: *Proceedings of the IEEE International Conference on Robotics and Automation*. Anchorage, Alaska, USA, May 2010, pp. 142–147. URL: http://www.mrt.kit.edu/z/publ/download/Moosmann_ICRA10.pdf.
- [MHB20] S. Muckenhuber, H. Holzer, and Z. Bockaj. “Automotive lidar modelling approach based on material properties and lidar capabilities”. In: *Sensors* 20.11 (2020), p. 3309.
- [MHG15] M. Menze, C. Heipke, and A. Geiger. “Joint 3D Estimation of Vehicles and Scene Flow”. In: *Proc. of the ISPRS Workshop on Image Sequence Analysis (ISA)*. 2015.
- [MHG18] M. Menze, C. Heipke, and A. Geiger. “Object Scene Flow”. In: *ISPRS Journal of Photogrammetry and Remote Sensing (JPRS)* (2018).
- [Mil+20] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. “NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis”. In: *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part I*. Ed. by A. Vedaldi, H. Bischof, T. Brox, and J. Frahm. Vol. 12346. Lecture Notes in Computer Science. Springer, 2020, pp. 405–421. DOI: [10.1007/978-3-030-58452-8_24](https://doi.org/10.1007/978-3-030-58452-8_24). URL: https://doi.org/10.1007/978-3-030-58452-8%5C_24.
- [MOH20] H. Mittal, B. Okorn, and D. Held. “Just Go With the Flow: Self-Supervised Scene Flow Estimation”. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*. IEEE, 2020, pp. 11174–11182. DOI: [10.1109/CVPR42600.2020.01119](https://doi.org/10.1109/CVPR42600.2020.01119). URL: <https://doi.org/10.1109/CVPR42600.2020.01119>.
- [Mun+22] J. Munkberg, J. Hasselgren, T. Shen, J. Gao, W. Chen, A. Evans, T. Müller, and S. Fidler. “Extracting Triangular 3D Models, Materials, and Lighting From Images”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2022, pp. 8280–8290.
- [Naj+22] M. Najibi, J. Ji, Y. Zhou, C. R. Qi, X. Yan, S. Ettinger, and D. Anguelov. “Motion Inspired Unsupervised Perception and Prediction in Autonomous Driving”. In: *Computer Vision - ECCV 2022 - 17th European Conference, Tel Aviv, Israel, October 23-27, 2022, Proceedings, Part XXXVIII*. Ed. by S. Avidan, G. J. Brostow, M. Cissé, G. M. Farinella, and T. Hassner. Vol. 13698. Lecture Notes in Computer Science. Springer, 2022, pp. 424–443. DOI: [10.1007/978-3-031-19839-7_25](https://doi.org/10.1007/978-3-031-19839-7_25). URL: https://doi.org/10.1007/978-3-031-19839-7%5C_25.

- [NFS15] R. A. Newcombe, D. Fox, and S. M. Seitz. “DynamicFusion: Reconstruction and tracking of non-rigid scenes in real-time”. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. IEEE Computer Society, 2015, pp. 343–352. DOI: [10.1109/CVPR.2015.7298631](https://doi.org/10.1109/CVPR.2015.7298631). URL: <https://doi.org/10.1109/CVPR.2015.7298631>.
- [Ngu+20] D. T. Nguyen, C. K. Mummadi, T. Ngo, T. H. P. Nguyen, L. Beggel, and T. Brox. “SELF: Learning to Filter Noisy Labels with Self-Ensembling”. In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL: <https://openreview.net/forum?id=HkgsPhNYPS>.
- [NIK23] K. Nakashima, Y. Iwashita, and R. Kurazume. “Generative Range Imaging for Learning Scene Priors of 3D LiDAR Data”. In: *IEEE/CVF Winter Conference on Applications of Computer Vision, WACV 2023, Waikoloa, HI, USA, January 2-7, 2023*. IEEE, 2023, pp. 1256–1266. DOI: [10.1109/WACV56688.2023.00131](https://doi.org/10.1109/WACV56688.2023.00131). URL: <https://doi.org/10.1109/WACV56688.2023.00131>.
- [NK21] K. Nakashima and R. Kurazume. “Learning to Drop Points for LiDAR Scan Synthesis”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2021, Prague, Czech Republic, September 27 - Oct. 1, 2021*. IEEE, 2021, pp. 222–229. DOI: [10.1109/IROS51168.2021.9636747](https://doi.org/10.1109/IROS51168.2021.9636747). URL: <https://doi.org/10.1109/IROS51168.2021.9636747>.
- [Pas+19] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Z. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. Ed. by H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. B. Fox, and R. Garnett. 2019, pp. 8024–8035. URL: <https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html>.
- [PBM20] G. Puy, A. Boulch, and R. Marlet. “FLOT: Scene Flow on Point Clouds Guided by Optimal Transport”. In: *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part XXVIII*. Ed. by A. Vedaldi, H. Bischof, T. Brox, and J. Frahm. Vol. 12373. Lecture Notes in Computer Science. Springer, 2020, pp. 527–544. DOI: [10.1007/978-3-030-58604-1_32](https://doi.org/10.1007/978-3-030-58604-1_32). URL: https://doi.org/10.1007/978-3-030-58604-1_32.

- [Ped+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.
- [PHL20] J. K. Pontes, J. Hays, and S. Lucey. “Scene Flow from Point Clouds with or without Learning”. In: *8th International Conference on 3D Vision, 3DV 2020, Virtual Event, Japan, November 25-28, 2020*. Ed. by V. Struc and F. G. Fernández. IEEE, 2020, pp. 261–270. DOI: [10.1109/3DV50981.2020.00036](https://doi.org/10.1109/3DV50981.2020.00036). URL: <https://doi.org/10.1109/3DV50981.2020.00036>.
- [PJH16] M. Pharr, W. Jakob, and G. Humphreys. *Physically Based Rendering: From Theory to Implementation*. 3rd. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2016. ISBN: 0128006455.
- [PKF05] J. Pons, R. Keriven, and O. D. Faugeras. “Modelling Dynamic Scenes by Registering Multi-View Image Sequences”. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2005), 20-26 June 2005, San Diego, CA, USA*. IEEE Computer Society, 2005, pp. 822–827. DOI: [10.1109/CVPR.2005.227](https://doi.org/10.1109/CVPR.2005.227). URL: <https://doi.org/10.1109/CVPR.2005.227>.
- [Pou+20] O. Poursaeed, T. Jiang, H. Qiao, N. Xu, and V. G. Kim. “Self-Supervised Learning of Point Clouds via Orientation Estimation”. In: *8th International Conference on 3D Vision, 3DV 2020, Virtual Event, Japan, November 25-28, 2020*. Ed. by V. Struc and F. G. Fernández. IEEE, 2020, pp. 1018–1028. DOI: [10.1109/3DV50981.2020.00112](https://doi.org/10.1109/3DV50981.2020.00112). URL: <https://doi.org/10.1109/3DV50981.2020.00112>.
- [Pre20] V. C. S. A. Pressezentrum. *Auf dem Weg zum autonomen Fahren: Nächste Volvo Modellgeneration nutzt Lidar-Technologie von Luminar*. 2020. URL: <https://www.media.volvocars.com/ch/de-ch/media/pressreleases/268323/auf-dem-weg-zum-autonomen-fahren-nachste-volvo-modellgeneration-nutzt-lidar-technologie-von-luminar> (visited on 05/26/2024).
- [Qi+17] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. “PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society, 2017, pp. 77–85. DOI: [10.1109/CVPR.2017.16](https://doi.org/10.1109/CVPR.2017.16). URL: <https://doi.org/10.1109/CVPR.2017.16>.
- [Rad+18] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, et al. “Improving language understanding by generative pre-training”. In: (2018).

- [Ran+19] A. Ranjan, V. Jampani, L. Balles, K. Kim, D. Sun, J. Wulff, and M. J. Black. “Competitive Collaboration: Joint Unsupervised Learning of Depth, Camera Motion, Optical Flow and Motion Segmentation”. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 2019, pp. 12240–12249. DOI: [10.1109/CVPR.2019.01252](https://doi.org/10.1109/CVPR.2019.01252). URL: http://openaccess.thecvf.com/content%5C_CVPR%5C_2019/html/Ranjan%5C_Competitive%5C_Collaboration%5C_Joint%5C_Unsupervised%5C_Learning%5C_of%5C_Depth%5C_Camera%5C_Motion%5C_Optical%5C_CVPR%5C_2019%5C_paper.html.
- [REG19] C. B. Rist, M. Enzweiler, and D. M. Gavrilă. “Cross-Sensor Deep Domain Adaptation for LiDAR Detection and Segmentation”. In: *2019 IEEE Intelligent Vehicles Symposium, IV 2019, Paris, France, June 9-12, 2019*. IEEE, 2019, pp. 1535–1542. DOI: [10.1109/IVS.2019.8814047](https://doi.org/10.1109/IVS.2019.8814047). URL: <https://doi.org/10.1109/IVS.2019.8814047>.
- [Ren+17] Z. Ren, J. Yan, B. Ni, B. Liu, X. Yang, and H. Zha. “Unsupervised Deep Learning for Optical Flow Estimation”. In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*. Ed. by S. Singh and S. Markovitch. AAAI Press, 2017, pp. 1495–1501. DOI: [10.1609/AAAI.V31I1.10723](https://doi.org/10.1609/AAAI.V31I1.10723). URL: <https://doi.org/10.1609/aaai.v31i1.10723>.
- [Ros+16] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. M. Lopez. “The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 3234–3243.
- [RPB12] A. Roncat, N. Pfeifer, and C. Briese. “A Linear Approach for Radiometric Calibration of Full-Waveform Lidar Data”. In: vol. 8537. Nov. 2012. DOI: [10.1117/12.970305](https://doi.org/10.1117/12.970305).
- [Sal+19] A. E. Sallab, I. Sobh, M. Zahran, and N. Essam. “LiDAR Sensor modeling and Data augmentation with GANs for Autonomous driving”. In: *CoRR* abs/1905.07290 (2019). arXiv: [1905.07290](https://arxiv.org/abs/1905.07290). URL: <http://arxiv.org/abs/1905.07290>.
- [SB12] E. Sun D. et al. and Sudderth and M. J. Black. “Layered segmentation and optical flow estimation over time”. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2012, pp. 1768–1775.
- [Sch94] C. Schlick. “An Inexpensive BRDF Model for Physically-Based Rendering”. In: *Comput. Graph. Forum* 13.3 (1994), pp. 233–246. DOI: [10.1111/1467-8659.1330233](https://doi.org/10.1111/1467-8659.1330233). URL: <https://doi.org/10.1111/1467-8659.1330233>.

- [Sed+24] B. Sedek, M. Zygmunt, M. Jakubaszek, T. Drozd, and J. Wojtanowski. “Real-Time Walk Error Compensation Method Using Echo Signal Magnitude Measurement in ToF Laser Scanners”. In: *Sensors* 24.3 (2024). ISSN: 1424-8220. DOI: [10.3390/s24030733](https://doi.org/10.3390/s24030733). URL: <https://www.mdpi.com/1424-8220/24/3/733>.
- [Sei+24] J. Seidenschwarz, A. Ošep, F. Ferroni, S. Lucey, and L. Leal-Taixé. *SeMoLi: What Moves Together Belongs Together*. 2024. arXiv: [2402.19463 \[cs.CV\]](https://arxiv.org/abs/2402.19463). URL: <https://arxiv.org/abs/2402.19463>.
- [Sem23] L. Semiconductor Components Industries. *Analytical Evaluation of Signal to Noise Ratios for SiPM and APD in dToF LiDAR Applications*. Tech. rep. AND90257/D. 2023.
- [Sha+17] S. Shah, D. Dey, C. Lovett, and A. Kapoor. “AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles”. In: *Field and Service Robotics*. 2017. eprint: [arXiv:1705.05065](https://arxiv.org/abs/1705.05065). URL: <https://arxiv.org/abs/1705.05065>.
- [Sha+18] L. Shao, P. Shah, V. Dwaracherla, and J. Bohg. “Motion-Based Object Segmentation Based on Dense RGB-D Scene Flow”. In: *IEEE Robotics Autom. Lett.* 3.4 (2018), pp. 3797–3804. DOI: [10.1109/LRA.2018.2856525](https://doi.org/10.1109/LRA.2018.2856525). URL: <https://doi.org/10.1109/LRA.2018.2856525>.
- [She+21] Z. Shen, H. Liang, L. Lin, Z. Wang, W. Huang, and J. Yu. “Fast Ground Segmentation for 3D LiDAR Point Cloud Based on Jump-Convolution-Process”. In: *Remote. Sens.* 13.16 (2021), p. 3239. DOI: [10.3390/rs13163239](https://doi.org/10.3390/rs13163239). URL: <https://doi.org/10.3390/rs13163239>.
- [Shi+20] S. Shi, C. Guo, L. Jiang, Z. Wang, J. Shi, X. Wang, and H. Li. “PV-RCNN: Point-Voxel Feature Set Abstraction for 3D Object Detection”. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*. Computer Vision Foundation / IEEE, 2020, pp. 10526–10535. DOI: [10.1109/CVPR42600.2020.01054](https://openaccess.thecvf.com/content%5C_CVPR%5C_2020/html/Shi%5C_PV-RCNN%5C_Point-Voxel%5C_Feature%5C_Set%5C_Abstraction%5C_for%5C_3D%5C_Object%5C_Detection%5C_CVPR%5C_2020%5C_paper.html). URL: https://openaccess.thecvf.com/content%5C_CVPR%5C_2020/html/Shi%5C_PV-RCNN%5C_Point-Voxel%5C_Feature%5C_Set%5C_Abstraction%5C_for%5C_3D%5C_Object%5C_Detection%5C_CVPR%5C_2020%5C_paper.html.
- [Shi+21] S. Shi, Z. Wang, J. Shi, X. Wang, and H. Li. “From Points to Parts: 3D Object Detection From Point Cloud With Part-Aware and Part-Aggregation Network”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 43.8 (2021), pp. 2647–2664. DOI: [10.1109/TPAMI.2020.2977026](https://doi.org/10.1109/TPAMI.2020.2977026). URL: <https://doi.org/10.1109/TPAMI.2020.2977026>.
- [Shi+22] S. Shin, S. Golodetz, M. Vankadari, K. Zhou, A. Markham, and N. Trigoni. “Sample, Crop, Track: Self-Supervised Mobile 3D Object Detection for Urban Driving LiDAR”. In: *CoRR* abs/2209.10471 (2022). DOI: [10.48550/](https://arxiv.org/abs/2209.10471)

- arXiv.2209.10471. arXiv: 2209.10471. URL: <https://doi.org/10.48550/arXiv.2209.10471>.
- [Sko01] M. Skolnik. *Introduction to Radar Systems*. Electrical engineering series. McGraw-Hill, 2001. ISBN: 9780071181891. URL: <https://books.google.de/books?id=Y6-APwAACAAJ>.
- [SMS15] N. Srivastava, E. Mansimov, and R. Salakhutdinov. “Unsupervised Learning of Video Representations using LSTMs”. In: *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*. Ed. by F. R. Bach and D. M. Blei. Vol. 37. JMLR Workshop and Conference Proceedings. JMLR.org, 2015, pp. 843–852. URL: <http://proceedings.mlr.press/v37/srivastava15.html>.
- [Son+24] Z. Song, Z. He, X. Li, Q. Ma, R. Ming, Z. Mao, H. Pei, L. Peng, J. Hu, D. Yao, and Y. Zhang. “Synthetic Datasets for Autonomous Driving: A Survey”. In: *IEEE Trans. Intell. Veh.* 9.1 (2024), pp. 1847–1864. DOI: [10.1109/TIV.2023.3331024](https://doi.org/10.1109/TIV.2023.3331024). URL: <https://doi.org/10.1109/TIV.2023.3331024>.
- [SS19] J. Sauder and B. Sievers. “Context Prediction for Unsupervised Deep Learning on Point Clouds”. In: *CoRR* abs/1901.08396 (2019). arXiv: [1901.08396](https://arxiv.org/abs/1901.08396). URL: <http://arxiv.org/abs/1901.08396>.
- [SSP15] D. Sun, E. B. Sudderth, and H. Pfister. “Layered RGBD scene flow estimation”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), pp. 548–556.
- [Su+15] H. Su, C. R. Qi, Y. Li, and L. J. Guibas. “Render for cnn: Viewpoint estimation in images using cnns trained with rendered 3d model views”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 2686–2694.
- [Sun+20] P. Sun, H. Kretschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine, V. Vasudevan, W. Han, J. Ngiam, H. Zhao, A. Timofeev, S. Ettinger, M. Krivokon, A. Gao, A. Joshi, Y. Zhang, J. Shlens, Z. Chen, and D. Anguelov. “Scalability in Perception for Autonomous Driving: Waymo Open Dataset”. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*. IEEE, 2020, pp. 2443–2451. DOI: [10.1109/CVPR42600.2020.00252](https://doi.org/10.1109/CVPR42600.2020.00252). URL: <https://doi.org/10.1109/CVPR42600.2020.00252>.
- [Sun+21] P. Sun, W. Wang, Y. Chai, G. Elsayed, A. Bewley, X. Zhang, C. Sminchisescu, and D. Anguelov. “RSN: Range Sparse Net for Efficient, Accurate LiDAR 3D Object Detection”. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*. Computer Vision Foundation / IEEE, 2021, pp. 5725–5734. DOI: [10.1109/CVPR46437.2021.00567](https://openaccess.thecvf.com/content/CVPR2021/html/Sun%5C_RSN%5C_Range%5C_Sparse%5C_). URL: https://openaccess.thecvf.com/content/CVPR2021/html/Sun%5C_RSN%5C_Range%5C_Sparse%5C_

Net%5C_for%5C_Efficient%5C_Accurate%5C_LiDAR%5C_3D%5C_Object%5C_CVPR%5C_2021%5C_paper.html.

- [Sun+22] T. Sun, M. Segù, J. Postels, Y. Wang, L. V. Gool, B. Schiele, F. Tombari, and F. Yu. “SHIFT: A Synthetic Driving Dataset for Continuous Multi-Task Domain Adaptation”. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*. IEEE, 2022, pp. 21339–21350. DOI: [10.1109/CVPR52688.2022.02068](https://doi.org/10.1109/CVPR52688.2022.02068). URL: <https://doi.org/10.1109/CVPR52688.2022.02068>.
- [SVL14] I. Sutskever, O. Vinyals, and Q. V. Le. “Sequence to Sequence Learning with Neural Networks”. In: *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger. 2014, pp. 3104–3112. URL: <https://proceedings.neurips.cc/paper/2014/hash/a14ac55a4f27472c5d894ec1c3c743d2-Abstract.html>.
- [SWL19] S. Shi, X. Wang, and H. Li. “PointRCNN: 3D Object Proposal Generation and Detection From Point Cloud”. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 2019, pp. 770–779. DOI: [10.1109/CVPR.2019.00086](https://doi.org/10.1109/CVPR.2019.00086). URL: http://openaccess.thecvf.com/content%5C_CVPR%5C_2019/html/Shi%5C_PointRCNN%5C_3D%5C_Object%5C_Proposal%5C_Generation%5C_and%5C_Detection%5C_From%5C_Point%5C_Cloud%5C_CVPR%5C_2019%5C_paper.html.
- [SY22] Z. Song and B. Yang. “OGC: Unsupervised 3D Object Segmentation from Rigid Dynamics of Point Clouds”. In: *Advances in Neural Information Processing Systems*. Ed. by A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho. 2022. URL: <https://openreview.net/forum?id=ecNbE00tqBU>.
- [TAG20] A. K. Thabet, H. Alwassel, and B. Ghanem. “Self-Supervised Learning of Local Features in 3D Point Clouds”. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR Workshops 2020, Seattle, WA, USA, June 14-19, 2020*. IEEE, 2020, pp. 4048–4052. DOI: [10.1109/CVPRW50498.2020.00477](https://doi.org/10.1109/CVPRW50498.2020.00477). URL: <https://doi.org/10.1109/CVPRW50498.2020.00477>.
- [TD20] Z. Teed and J. Deng. “RAFT: Recurrent All-Pairs Field Transforms for Optical Flow”. In: *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part II*. Ed. by A. Vedaldi, H. Bischof, T. Brox, and J. Frahm. Vol. 12347. Lecture Notes in Computer Science. Springer, 2020, pp. 402–419. DOI: [10.1007/978-3-030-58536-5_24](https://doi.org/10.1007/978-3-030-58536-5_24). URL: https://doi.org/10.1007/978-3-030-58536-5_24.

- [Tea20] O. D. Team. *OpenPCDet: An Open-source Toolbox for 3D Object Detection from Point Clouds*. <https://github.com/open-mmlab/OpenPCDet>. 2020.
- [Thé+22] R. Théodose, D. Denis, T. Chateau, V. Frémont, and P. Checchin. “A Deep Learning Approach for LiDAR Resolution-Agnostic Object Detection”. In: *IEEE Trans. Intell. Transp. Syst.* 23.9 (2022), pp. 14582–14593. DOI: [10.1109/TITS.2021.3130487](https://doi.org/10.1109/TITS.2021.3130487). URL: <https://doi.org/10.1109/TITS.2021.3130487>.
- [Tho+20] H. Thomas, B. Agro, M. Gridseth, J. Zhang, and T. D. Barfoot. “Self-Supervised Learning of Lidar Segmentation for Autonomous Indoor Navigation”. In: *CoRR* abs/2012.05897 (2020). arXiv: [2012.05897](https://arxiv.org/abs/2012.05897). URL: <https://arxiv.org/abs/2012.05897>.
- [Tis+20] I. Tishchenko, S. Lombardi, M. R. Oswald, and M. Pollefeys. “Self-Supervised Learning of Non-Rigid Residual Flow and Ego-Motion”. In: *8th International Conference on 3D Vision, 3DV 2020, Virtual Event, Japan, November 25-28, 2020*. Ed. by V. Struc and F. G. Fernández. IEEE, 2020, pp. 150–159. DOI: [10.1109/3DV50981.2020.00025](https://doi.org/10.1109/3DV50981.2020.00025). URL: <https://doi.org/10.1109/3DV50981.2020.00025>.
- [Ton+24] A. Tonderski, C. Lindström, G. Hess, W. Ljungbergh, L. Svensson, and C. Petersson. “NeuRAD: Neural Rendering for Autonomous Driving”. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2024, Seattle, WA, USA, June 16-22, 2024*. IEEE, 2024, pp. 14895–14904. DOI: [10.1109/CVPR52733.2024.01411](https://doi.org/10.1109/CVPR52733.2024.01411). URL: <https://doi.org/10.1109/CVPR52733.2024.01411>.
- [TPP14] B. Träuble, S. Pauen, and D. Poulin-Dubois. “Speed and direction changes induce the perception of animacy in 7-month-old infants”. In: *Frontiers in Psychology* 5 (2014). ISSN: 1664-1078. DOI: [10.3389/fpsyg.2014.01141](https://doi.org/10.3389/fpsyg.2014.01141). URL: <https://www.frontiersin.org/articles/10.3389/fpsyg.2014.01141>.
- [Tri+20] L. T. Triess, D. Peter, C. B. Rist, and J. M. Zöllner. “Scan-based Semantic Segmentation of LiDAR Point Clouds: An Experimental Study”. In: *IEEE Intelligent Vehicles Symposium, IV 2020, Las Vegas, NV, USA, October 19 - November 13, 2020*. IEEE, 2020, pp. 1116–1121. DOI: [10.1109/IV47402.2020.9304631](https://doi.org/10.1109/IV47402.2020.9304631). URL: <https://doi.org/10.1109/IV47402.2020.9304631>.
- [Tri+21] L. T. Triess, D. Peter, S. A. Baur, and J. M. Zöllner. “Quantifying Point Cloud Realism Through Adversarially Learned Latent Representations”. In: *Pattern Recognition - 43rd DAGM German Conference, DAGM GCPR 2021, Bonn, Germany, September 28 - October 1, 2021, Proceedings*. Ed. by C. Bauckhage, J. Gall, and A. G. Schwing. Vol. 13024. Lecture Notes in Computer Science. Springer, 2021, pp. 681–696. DOI: [10.1007/978-3-](https://doi.org/10.1007/978-3-)

030-92659-5_44. URL: https://doi.org/10.1007/978-3-030-92659-5%5C_44.

- [Tun+17] H. Tung, H. Tung, E. Yumer, and K. Fragkiadaki. “Self-supervised Learning of Motion Capture”. In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. Ed. by I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett. 2017, pp. 5236–5246. URL: <https://proceedings.neurips.cc/paper/2017/hash/ab452534c5ce28c4fbb0e102d4a4fb2e-Abstract.html>.
- [Ved+99] S. Vedula, S. Baker, P. Rander, R. T. Collins, and T. Kanade. “Three-Dimensional Scene Flow”. In: *Proceedings of the International Conference on Computer Vision, Kerkyra, Corfu, Greece, September 20-25, 1999*. IEEE Computer Society, 1999, pp. 722–729. DOI: [10.1109/ICCV.1999.790293](https://doi.org/10.1109/ICCV.1999.790293). URL: <https://doi.org/10.1109/ICCV.1999.790293>.
- [Vel13] I. Velodyne. *USER’S MANUAL AND PROGRAMMING GUIDE HDL-64E S3 High Definition LiDAR Sensor*. English. Velodyne, Inc. 2013. 54 pp.
- [Viz+23] I. Vizzo, T. Guadagnino, B. Mersch, L. Wiesmann, J. Behley, and C. Stachniss. “KISS-ICP: In Defense of Point-to-Point ICP - Simple, Accurate, and Robust Registration If Done the Right Way”. In: *IEEE Robotics Autom. Lett.* 8.2 (2023), pp. 1029–1036. DOI: [10.1109/LRA.2023.3236571](https://doi.org/10.1109/LRA.2023.3236571). URL: <https://doi.org/10.1109/LRA.2023.3236571>.
- [Vog12] M. Vogel. “Principles of Lasers, 5th edn., by O. Svelto”. In: *Contemporary physics* 53.2 (2012), pp. 173–173. ISSN: 1366-5812. DOI: [10.1080/00107514.2011.647714](https://doi.org/10.1080/00107514.2011.647714). URL: <https://repository.gsi.de/record/206584>.
- [Vri24] J. de Vries. *Physically Based Rendering*. Accessed: 2024-11-23. 2024. URL: <https://learnopengl.com/PBR/Theory>.
- [VSR15] C. Vogel, K. Schindler, and S. Roth. “3D Scene Flow Estimation with a Piecewise Rigid Scene Model”. In: *Int. J. Comput. Vis.* 115.1 (2015), pp. 1–28. DOI: [10.1007/s11263-015-0806-0](https://doi.org/10.1007/s11263-015-0806-0). URL: <https://doi.org/10.1007/s11263-015-0806-0>.
- [Wag10] W. Wagner. “Radiometric calibration of small-footprint full-waveform airborne laser scanner measurements: Basic physical concepts”. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 65.6 (2010), pp. 505–513. ISSN: 09242716. DOI: [10.1016/j.isprsjprs.2010.06.007](https://doi.org/10.1016/j.isprsjprs.2010.06.007). URL: <http://dx.doi.org/10.1016/j.isprsjprs.2010.06.007>.

- [Wan+18a] S. Wang, S. Suo, W. Ma, A. Pokrovsky, and R. Urtasun. “Deep Parametric Continuous Convolutional Neural Networks”. In: *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. IEEE Computer Society, 2018, pp. 2589–2597. DOI: [10.1109/CVPR.2018.00274](https://doi.org/10.1109/CVPR.2018.00274). URL: http://openaccess.thecvf.com/content%5C_cvpr%5C_2018/html/Wang%5C_Deep%5C_Parametric%5C_Continuous%5C_CVPR%5C_2018%5C_paper.html.
- [Wan+18b] Y. Wang, Y. Yang, Z. Yang, L. Zhao, P. Wang, and W. Xu. “Occlusion Aware Unsupervised Learning of Optical Flow”. In: *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. Computer Vision Foundation / IEEE Computer Society, 2018, pp. 4884–4893. DOI: [10.1109/CVPR.2018.00513](https://doi.org/10.1109/CVPR.2018.00513). URL: http://openaccess.thecvf.com/content%5C_cvpr%5C_2018/html/Wang%5C_Occlusion%5C_Aware%5C_Unsupervised%5C_CVPR%5C_2018%5C_paper.html.
- [Wan+20a] G. Wang, X. Wu, Z. Liu, and H. Wang. “Hierarchical Attention Learning of Scene Flow in 3D Point Clouds”. In: *CoRR abs/2010.05762 (2020)*. arXiv: [2010.05762](https://arxiv.org/abs/2010.05762). URL: <https://arxiv.org/abs/2010.05762>.
- [Wan+20b] Y. Wang, X. Chen, Y. You, L. E. Li, B. Hariharan, M. E. Campbell, K. Q. Weinberger, and W. Chao. “Train in Germany, Test in the USA: Making 3D Object Detectors Generalize”. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*. Computer Vision Foundation / IEEE, 2020, pp. 11710–11720. DOI: [10.1109/CVPR42600.2020.01173](https://doi.org/10.1109/CVPR42600.2020.01173). URL: https://openaccess.thecvf.com/content%5C_CVPR%5C_2020/html/Wang%5C_Train%5C_in%5C_Germany%5C_Test%5C_in%5C_the%5C_USA%5C_Making%5C_3D%5C_Object%5C_CVPR%5C_2020%5C_paper.html.
- [Wan+20c] Z. Wang, S. Li, H. Howard-Jenkins, V. A. Prisacariu, and M. Chen. “FlowNet3D++: Geometric Losses For Deep Scene Flow Estimation”. In: *IEEE Winter Conference on Applications of Computer Vision, WACV 2020, Snowmass Village, CO, USA, March 1-5, 2020*. IEEE, 2020, pp. 91–98. DOI: [10.1109/WACV45572.2020.9093302](https://doi.org/10.1109/WACV45572.2020.9093302). URL: <https://doi.org/10.1109/WACV45572.2020.9093302>.
- [Way24] Waymo LLC. *WAYMO One is now open to everyone in San Francisco*. <https://waymo.com/blog/2024/06/waymo-one-is-now-open-to-everyone-in-san-francisco/>. Accessed: 2024-07-14. 2024.
- [WB14] J. Wulff and M. J. Black. “Modeling Blurred Video with Layers”. In: *Computer Vision – ECCV 2014*. Vol. 8694. Lecture Notes in Computer Science. Springer International Publishing, Sept. 2014, pp. 236–252. DOI: [10.1007/978-3-319-10599-4_16](https://doi.org/10.1007/978-3-319-10599-4_16).

- [WCZ22] Y. Wang, Y. Chen, and Z. Zhang. “4D Unsupervised Object Discovery”. In: *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*. Ed. by S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh. 2022. URL: http://papers.nips.cc/paper%5C_files/paper/2022/hash/e7407ab5e89c405d28ff6807ffec594a-Abstract-Conference.html.
- [Wed+08] A. Wedel, C. Rabe, T. Vaudrey, T. Brox, U. Franke, and D. Cremers. “Efficient Dense Scene Flow from Sparse or Dense Stereo Data”. In: *Computer Vision - ECCV 2008, 10th European Conference on Computer Vision, Marseille, France, October 12-18, 2008, Proceedings, Part I*. Ed. by D. A. Forsyth, P. H. S. Torr, and A. Zisserman. Vol. 5302. Lecture Notes in Computer Science. Springer, 2008, pp. 739–751. DOI: [10.1007/978-3-540-88682-2_56](https://doi.org/10.1007/978-3-540-88682-2_56). URL: https://doi.org/10.1007/978-3-540-88682-2_56.
- [Wei+20] Y. Wei, Z. Wang, Y. Rao, J. Lu, and J. Zhou. “PV-RAFT: Point-Voxel Correlation Fields for Scene Flow Estimation of Point Clouds”. In: *CoRR* abs/2012.00987 (2020). arXiv: [2012.00987](https://arxiv.org/abs/2012.00987). URL: <https://arxiv.org/abs/2012.00987>.
- [WG15] X. Wang and A. Gupta. “Unsupervised Learning of Visual Representations Using Videos”. In: *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*. IEEE Computer Society, 2015, pp. 2794–2802. DOI: [10.1109/ICCV.2015.320](https://doi.org/10.1109/ICCV.2015.320). URL: <https://doi.org/10.1109/ICCV.2015.320>.
- [Wil+21] B. Wilson, W. Qi, T. Agarwal, J. Lambert, J. Singh, S. Khandelwal, B. Pan, R. Kumar, A. Hartnett, J. K. Pontes, D. Ramanan, P. Carr, and J. Hays. “Argoverse 2: Next Generation Datasets for Self-driving Perception and Forecasting”. In: *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks (NeurIPS Datasets and Benchmarks 2021)*. 2021.
- [Wil17] G. M. Williams. “Optimization of eyesafe avalanche photodiode lidar for automobile safety and autonomous navigation systems”. In: *Optical Engineering* 56, 031224 (Mar. 2017), p. 031224. DOI: [10.1117/1.OE.56.3.031224](https://doi.org/10.1117/1.OE.56.3.031224).
- [Wil18] G. Williams. *Range-walk correction using time over threshold*. Tech. rep. 2018.
- [WJE19] X. Wang, A. Jabri, and A. A. Efros. “Learning Correspondence From the Cycle-Consistency of Time”. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 2019, pp. 2566–2576. DOI: [10.1109/CVPR.2019.00267](https://doi.org/10.1109/CVPR.2019.00267). URL: <http://openaccess.thecvf>.

- [com/content%5C_CVPR%5C_2019/html/Wang%5C_Learning%5C_Correspondence%5C_From%5C_the%5C_Cycle-Consistency%5C_of%5C_Time%5C_CVPR%5C_2019%5C_paper.html](https://arxiv.org/abs/1904.02680).
- [Woj+14] J. Wojtanowski, M. Zygmunt, M. Kaszczuk, Z. Mierczyk, and M. Muzal. “Comparison of 905 nm and 1550 nm semiconductor laser rangefinders’ performance deterioration due to adverse environmental conditions”. In: *Opto-Electronics Review* 22 (June 2014). DOI: [10.2478/s11772-014-0190-2](https://doi.org/10.2478/s11772-014-0190-2).
- [Woz+24] M. K. Wozniak, M. Hansson, M. Thiel, and P. Jensfelt. “UADA3D: Un-supervised Adversarial Domain Adaptation for 3D Object Detection with Sparse LiDAR and Large Domain Gaps”. In: *CoRR* abs/2403.17633 (2024). DOI: [10.48550/ARXIV.2403.17633](https://doi.org/10.48550/ARXIV.2403.17633). arXiv: [2403.17633](https://arxiv.org/abs/2403.17633). URL: <https://doi.org/10.48550/arXiv.2403.17633>.
- [Wu+20] W. Wu, Z. Wang, Z. Li, W. Liu, and F. Li. “PointPWC-Net: Cost Volume on Point Clouds for (Self-)Supervised Scene Flow Estimation”. In: *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part V*. Ed. by A. Vedaldi, H. Bischof, T. Brox, and J. Frahm. Vol. 12350. Lecture Notes in Computer Science. Springer, 2020, pp. 88–107. DOI: [10.1007/978-3-030-58558-7_6](https://doi.org/10.1007/978-3-030-58558-7_6). URL: https://doi.org/10.1007/978-3-030-58558-7_6.
- [Xie+20] S. Xie, J. Gu, D. Guo, C. R. Qi, L. J. Guibas, and O. Litany. “PointContrast: Unsupervised Pre-training for 3D Point Cloud Understanding”. In: *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part III*. Ed. by A. Vedaldi, H. Bischof, T. Brox, and J. Frahm. Vol. 12348. Lecture Notes in Computer Science. Springer, 2020, pp. 574–591. DOI: [10.1007/978-3-030-58580-8_34](https://doi.org/10.1007/978-3-030-58580-8_34). URL: https://doi.org/10.1007/978-3-030-58580-8_34.
- [Xu+21] Q. Xu, Y. Zhou, W. Wang, C. R. Qi, and D. Anguelov. “SPG: Unsupervised Domain Adaptation for 3D Object Detection via Semantic Point Generation”. In: *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*. IEEE, 2021, pp. 15426–15436. DOI: [10.1109/ICCV48922.2021.01516](https://doi.org/10.1109/ICCV48922.2021.01516). URL: <https://doi.org/10.1109/ICCV48922.2021.01516>.
- [Xu+22] R. Xu, H. Xiang, X. Xia, X. Han, J. Li, and J. Ma. “OPV2V: An Open Benchmark Dataset and Fusion Pipeline for Perception with Vehicle-to-Vehicle Communication”. In: *2022 International Conference on Robotics and Automation, ICRA 2022, Philadelphia, PA, USA, May 23-27, 2022*. IEEE, 2022, pp. 2583–2589. DOI: [10.1109/ICRA46639.2022.9812038](https://doi.org/10.1109/ICRA46639.2022.9812038). URL: <https://doi.org/10.1109/ICRA46639.2022.9812038>.

- [XW23] J. Xu and S. L. Waslander. “HyperMODEST: Self-Supervised 3D Object Detection with Confidence Score Filtering”. In: *CoRR* abs/2304.14446 (2023). DOI: [10.48550/arXiv.2304.14446](https://doi.org/10.48550/arXiv.2304.14446). arXiv: [2304.14446](https://arxiv.org/abs/2304.14446). URL: <https://doi.org/10.48550/arXiv.2304.14446>.
- [Yan+19] J. Yang, X. Liu, G. Gu, W. Qian, K. Ren, J. Zhang, P. Guo, Q. Gao, and Q. Chen. “A double threshold correction method for walk error in pulsed laser ranging system”. In: *Infrared Physics & Technology* 100 (2019), pp. 28–36. ISSN: 1350-4495. DOI: <https://doi.org/10.1016/j.infrared.2019.03.023>. URL: <https://www.sciencedirect.com/science/article/pii/S1350449518307801>.
- [Yan+20] Z. Yang, Y. Sun, S. Liu, and J. Jia. “3DSSD: Point-Based 3D Single Stage Object Detector”. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*. Computer Vision Foundation / IEEE, 2020, pp. 11037–11045. DOI: [10.1109/CVPR42600.2020.01105](https://openaccess.thecvf.com/content%5C_CVPR%5C_2020/html/Yang%5C_3DSSD%5C_Point-Based%5C_3D%5C_Single%5C_Stage%5C_Object%5C_Detector%5C_CVPR%5C_2020%5C_paper.html). URL: https://openaccess.thecvf.com/content%5C_CVPR%5C_2020/html/Yang%5C_3DSSD%5C_Point-Based%5C_3D%5C_Single%5C_Stage%5C_Object%5C_Detector%5C_CVPR%5C_2020%5C_paper.html.
- [YHD16] J. J. Yu, A. W. Harley, and K. G. Derpanis. “Back to Basics: Unsupervised Learning of Optical Flow via Brightness Constancy and Motion Smoothness”. In: *Computer Vision - ECCV 2016 Workshops - Amsterdam, The Netherlands, October 8-10 and 15-16, 2016, Proceedings, Part III*. Ed. by G. Hua and H. Jégou. Vol. 9915. Lecture Notes in Computer Science. 2016, pp. 3–10. DOI: [10.1007/978-3-319-49409-8_1](https://doi.org/10.1007/978-3-319-49409-8_1). URL: https://doi.org/10.1007/978-3-319-49409-8_1.
- [You+22] Y. You, K. Luo, C. P. Phoo, W. Chao, W. Sun, B. Hariharan, M. E. Campbell, and K. Q. Weinberger. “Learning to Detect Mobile Objects from LiDAR Scans Without Labels”. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*. IEEE, 2022, pp. 1120–1130. DOI: [10.1109/CVPR52688.2022.00120](https://doi.org/10.1109/CVPR52688.2022.00120). URL: <https://doi.org/10.1109/CVPR52688.2022.00120>.
- [YS18] Z. Yin and J. Shi. “GeoNet: Unsupervised Learning of Dense Depth, Optical Flow and Camera Pose”. In: *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. IEEE Computer Society, 2018, pp. 1983–1992. DOI: [10.1109/CVPR.2018.00212](http://openaccess.thecvf.com/content%5C_cvpr%5C_2018/html/Yin%5C_GeoNet%5C_Unsupervised%5C_Learning%5C_CVPR%5C_2018%5C_paper.html). URL: http://openaccess.thecvf.com/content%5C_cvpr%5C_2018/html/Yin%5C_GeoNet%5C_Unsupervised%5C_Learning%5C_CVPR%5C_2018%5C_paper.html.
- [YZK21] T. Yin, X. Zhou, and P. Krähenbühl. “Center-Based 3D Object Detection and Tracking”. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*. Computer Vision Foundation / IEEE, 2021, pp. 11784–11793. DOI: [10.1109/CVPR46437.2021.01161](https://doi.org/10.1109/CVPR46437.2021.01161).

- URL: https://openaccess.thecvf.com/content/CVPR2021/html/Yin%5C_Center-Based%5C_3D%5C_Object%5C_Detection%5C_and%5C_Tracking%5C_CVPR%5C_2021%5C_paper.html.
- [Zha+21] Z. Zhang, R. Girdhar, A. Joulin, and I. Misra. “Self-Supervised Pretraining of 3D Features on any Point-Cloud”. In: *CoRR* abs/2101.02691 (2021). arXiv: 2101.02691. URL: <https://arxiv.org/abs/2101.02691>.
- [Zha+22] Y. Zhang, Q. Hu, G. Xu, Y. Ma, J. Wan, and Y. Guo. “Not All Points Are Equal: Learning Highly Efficient Point-based Detectors for 3D LiDAR Point Clouds”. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*. IEEE, 2022, pp. 18931–18940. DOI: 10.1109/CVPR52688.2022.01838. URL: <https://doi.org/10.1109/CVPR52688.2022.01838>.
- [Zha+23] L. Zhang, A. J. Yang, Y. Xiong, S. Casas, B. Yang, M. Ren, and R. Urtasun. “Towards Unsupervised Object Detection From LiDAR Point Clouds”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2023, pp. 9317–9328.
- [Zha+24] J. Zhang, F. Zhang, S. Kuang, and L. Zhang. “NeRF-LiDAR: Generating Realistic LiDAR Point Clouds with Neural Radiance Fields”. In: *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2024, February 20-27, 2024, Vancouver, Canada*. Ed. by M. J. Wooldridge, J. G. Dy, and S. Natarajan. AAAI Press, 2024, pp. 7178–7186. DOI: 10.1609/AAAI.V38I7.28546. URL: <https://doi.org/10.1609/aaai.v38i7.28546>.
- [ZIE16] R. Zhang, P. Isola, and A. A. Efros. “Colorful Image Colorization”. In: *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part III*. Ed. by B. Leibe, J. Matas, N. Sebe, and M. Welling. Vol. 9907. Lecture Notes in Computer Science. Springer, 2016, pp. 649–666. DOI: 10.1007/978-3-319-46487-9_40. URL: https://doi.org/10.1007/978-3-319-46487-9_40.
- [ZLH18] Y. Zou, Z. Luo, and J. Huang. “DF-Net: Unsupervised Joint Learning of Depth and Flow Using Cross-Task Consistency”. In: *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part V*. Ed. by V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss. Vol. 11209. Lecture Notes in Computer Science. Springer, 2018, pp. 38–55. DOI: 10.1007/978-3-030-01228-1_3. URL: https://doi.org/10.1007/978-3-030-01228-1_3.

- [ZT18] Y. Zhou and O. Tuzel. “VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection”. In: *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. IEEE Computer Society, 2018, pp. 4490–4499. DOI: [10.1109/CVPR.2018.00472](https://doi.org/10.1109/CVPR.2018.00472). URL: http://openaccess.thecvf.com/content%5C_cvpr%5C_2018/html/Zhou%5C_VoxelNet%5C_End-to-End%5C_Learning%5C_CVPR%5C_2018%5C_paper.html.
- [Zua+20] V. Zuanazzi, J. van Vugt, O. Booiij, and P. Mettes. “Adversarial Self-Supervised Scene Flow Estimation”. In: *8th International Conference on 3D Vision, 3DV 2020, Virtual Event, Japan, November 25-28, 2020*. Ed. by V. Struc and F. G. Fernández. IEEE, 2020, pp. 1049–1058. DOI: [10.1109/3DV50981.2020.00115](https://doi.org/10.1109/3DV50981.2020.00115). URL: <https://doi.org/10.1109/3DV50981.2020.00115>.